In Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC 2002), Edinburgh, Scotland.

A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids

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

Abstract

Peer-to-peer (P2P) computing grids consist of peer nodes that communicate directly among themselves through wide-area networks and can act as both clients and servers. These systems have drawn much research attention since they promote Internet-scale resource and service sharing without any administration cost or centralized infrastructure support. However, aggregating different application services into a high-performance distributed application delivery in such systems is challenging due to the presence of dynamic performance information, arbitrary peer arrivals/departures, and systems' scalability requirement. In this paper, we propose a scalable QoS-aware service aggregation model to address the challenges. The model includes two tiers: (1) on-demand service composition tier, which is responsible for choosing and composing different application services into a service path satisfying the user's quality requirements; and (2) dynamic peer selection tier, which decides the specific peers where the chosen services are actually instantiated based on the dynamic, composite and distributed performance information. The model is designed and implemented in a fully distributed and self-organizing fashion. Finally, we show that the proposed model and algorithms can achieve better performance than common heuristic algorithms using large-scale simulations.

1 Introduction

In peer-to-peer (P2P) systems, computers, which are called "peers", communicate directly among themselves and can act as both clients and servers. Unlike conventional client-server systems, P2P systems are distributed systems without any centralized infrastructure or hierarchical organization. Peers voluntarily participate in utilizing or contributing resources and services in the system. Recently, P2P systems have drawn much research attention with the popularity of P2P file sharing systems such as Gnutella [1] and Napster [2]. P2P systems are attractive since they promote *resource sharing* such as the sharing of processing cycles and disk storage, *without any administration cost or centralized infrastructure support*. The previous research work on P2P systems has mainly focused on providing scalable P2P discovery services [7, 20, 16], which enable *data and resource sharing*, in P2P systems. However, few have addressed the *service sharing* problem which is important for the user of P2P systems to utilize a wealth of application services provided by other peers. Hence, we propose a scalable service aggregation model for P2P systems to automatically aggregate services into a high performance distributed application delivery with quality-of-service (QoS) guarantees to fulfill the user's requirements.

The problem of service instantiation and/or composition has been addressed by much research work under different context [8, 9, 22, 5]. However, most of the proposed approaches present the following major limitations when applying to P2P systems. First, they lack generic QoS support for coordinating arbitrary interactions between service instances. Second, they do not provide dynamic peer selection scheme since all services are assumed to be provided by dedicated servers. Third, they often assume a global view of the entire system in terms of performance information, which, however, is impossible in P2P systems due to the scalability requirement. Fourth, they do not consider the dynamic topological variation caused by arbitrary peer arrivals/departures in P2P systems.

In this paper, we address the above challenges by proposing a *scalable QoS-aware service aggregation model*, which is designed and implemented in a fully distributed and self-organizing fashion. The model in-

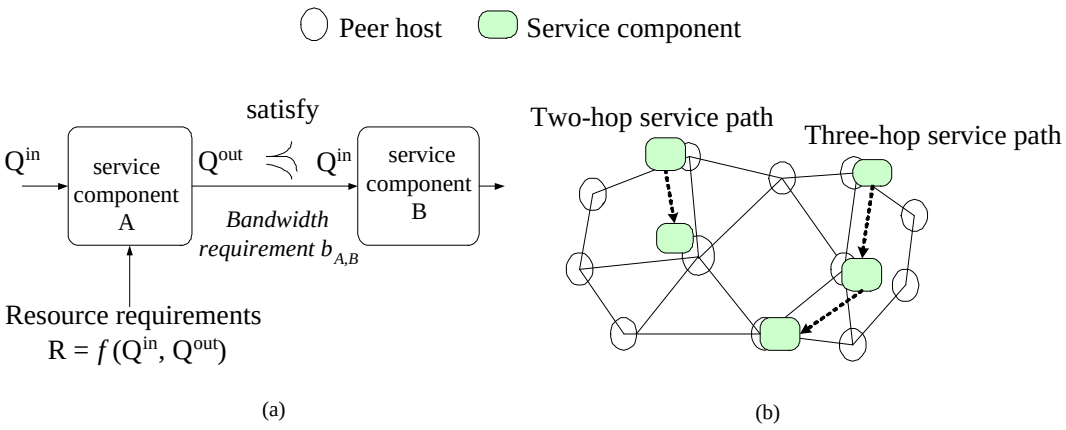

Figure 1. Illustration of the application service model for P2P systems.

cludes two tiers: (1) *on-demand service composition*; and (2) *dynamic peer selection*. The former is responsible for choosing suitable service instances, discovered in the current P2P system, to compose a distributed application delivery satisfying both the functional and non-functional (i.e., QoS) requirements of the user. To provide QoS support, we propose a quality consistency check algorithm to guarantee the consistency of quality parameters between any two interacting service instances while meeting the user's end-to-end QoS requirements. However, due to the inherent *redundancy* property of P2P systems, a service instance can have multiple replications, which are provided by different peers. Hence, we employ the *dynamic peer selection* tier to decide the specific peers where the service instances are actually executed. The selection decisions are made based on the dynamic and hop-by-hop [1] performance information such as system load, network bandwidth and delay. For providing highly available distributed applications in P2P systems, we also need to consider the topological variation, caused by arbitrary peer arrivals/departures, when we select peers for instantiating services. Hence, we use the *peer's uptime* as one of peer selection metrics, which is defined as the duration that the peer has remained connected to the P2P system. The up-to-date performance information is maintained at each peer host through a controlled, benefit-based probing method. The major advantages achieved by the *dynamic peer selection* are: (1) improved service request success rate; (2) improved service availability; (3) load balance in heterogenous environments; and (4) improved tolerance to the topological variation of P2P systems.

The rest of the paper is organized as follows. We introduce the system model in Section 2. Section 3 presents the design and algorithms of the scalable QoS-aware service aggregation model, followed by the extensive simulation results, presented in Section 4. We discuss related work in Section 5. Finally, we conclude this paper in Section 6.

[1]The hop here refers to the application-level (e.g., TCP or UDP) connection between two peers.

2 System Model

This section describes the system model. First, we present a component-based application service model to characterize the quality sensitive distributed applications such as the *video-on-demand* and *content retrieval* applications. Then, we introduce a network model for Internet-scale P2P systems. Finally, we present the overview of the QoS-aware service aggregation model.

2.1 QoS-Aware Application Service Model

We assume that each distributed application delivery consists of a list of composable service components, which are connected into a *service path*. Each service component accepts input data with a QoS level Q^{in} and generates output with a QoS level Q^{out}, both of which are vectors of application-level QoS parameters. The application-level QoS parameters can be data format (e.g., MPEG, JPEG) , frame rate (e.g., [0,20]fps), and others. In order to process input and generate output, a specific amount of resources R is required, which is a vector of required *end-system* resources (e.g., cpu, memory). The network resource requirements, such as bandwidth $b_{i,j}$, are associated with edges between two communicating components i and j. Figure 1(a) illustrates such a characterization in terms of QoS parameters and resources. Formally, we define the vectors Q^{in}, Q^{out}, and R as follows: $Q^{in} = [q_1^{in}, q_2^{in}, ..., q_n^{in}]$, $Q^{out} = [q_1^{out}, q_2^{out}, ..., q_n^{out}]$, $R = [r_1, r_2, ..., r_m]$. Intuitively, if a component A is connected to a component B,

the output QoS of A (Q_A^{out}) must "match" the input QoS requirements of component B (Q_B^{in}). In order to formally describe this *QoS consistency* requirements, we define an inter-component relation "$\preceq$", called "*satisfy*", as follows: $Q_A^{out} \preceq Q_B^{in}$ if and only if

$$\forall i, 1 \leq i \leq Dim(Q_B^{in}),\ \exists j, 1 \leq j \leq Dim(Q_A^{out}),$$
$$q_{Aj}^{out} = q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ single\ value;$$
$$q_{Aj}^{out} \subseteq q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ range\ value. \quad (1)$$

The "Dim(Q)" represents the dimension of the vector "Q". The *single value* QoS parameters include data format, resolution, and others. The *range value* QoS parameters include frame rate ([10fps,30fps]) and others. If the application delivery involves n peers except the peer requesting the application, then such an application delivery is defined as an n *hop service aggregation*, illustrated in Figure 1 (b). For example, the content retrieval application represents a single hop service aggregation. Note that the hop count here represents the number of application-level connections. Each hop in the service aggregation may include many network-level hops depending on the network distance between two peers.

2.2 Network Model

According to [17], peers are highly heterogeneous and also reluctant to report their performance information to other peers or deliberately misreport the information. Hence, in order to provide efficient QoS support, we assume that each peer proactively measures the performance information of other peers through probing. In order to achieve scalability and avoid flooding of probing messages, we assume that each peer can only probe a small number of "peer neighbors" whose resource information is the most important. To be specific, if peer B provides services that peer A needs, B is regarded as A's neighbor. If the service that peer B provides is the ith hop from the reverse direction of the service aggregation flow, then B is defined as A's *i-hop neighbor*. Moreover, if the service that peer B provides is part of an application that A needs, B is defined as the A's "direct neighbor". Otherwise, B is called "indirect neighbor". For example, in Figure 2, B_1, B_2, and B_3 are A's 1-hop *direct* neighbors. C_1 and D_1 are A's 2-hop and 3-hop *direct* neighbors, respectively. C_1 and D_1 are B_3's 1-hop and 2-hop *indirect* neighbors, respectively. Such a relationship is dynamically decided by our *neighbor resolution protocol* introduced later in Section 3.3. We define M as the maximum number of peers whose performance information is maintained by any peer. Under such a constraint, any peer first probes its 1-hop direct neighbors, then 1-hop indirect neighbors, then 2-hop direct neighbors and so on.

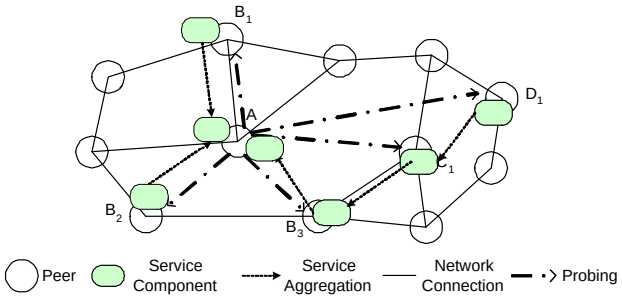

Figure 2. Illustration of the network model for P2P systems.

2.3 Service Aggregation Model

The service aggregation model dynamically composes and delivers high performance distributed applications for the user of P2P systems by exploiting the systems' inherent *redundancy* property, which is represented by the fact that: (1) the same application service (e.g., video player) can have multiple service instances (e.g., real player, windows media player), each of which has different Q^{in} and Q^{out} parameters; and (2) the same service instance (e.g., real player) can have multiple copies on different physical peer hosts (e.g., boston.cs.uiuc.edu, spica.hpl.hp.com). Hence, our solution tackles two key problems for aggregating services at setup time of each application session: (1) how to choose service instances with proper quality parameters Q^{in} and Q^{out} according to the user's end-to-end QoS requirements [2]; and (2) how to select proper peers to execute the chosen service instances, according to the dynamic performance information of all candidate peers. To address the above problems, the service aggregation model includes two cooperating tiers: (1) *on-demand service composition*; and (2) *dynamic peer selection*. Upon receiving a user request, the *on-demand service composition* tier first chooses among those candidate service instances with different Q^{in} and Q^{out} parameters, to establish a QoS consistent (equation(1), Section 2.1) service path satisfying the user's end-to-end QoS requirements. Next, the *dynamic peer selection* tier is responsible for selecting proper peers to execute the service instances chosen by the first tier, according to the dynamic and distributed performance information. Such a service aggregation model would be beneficial to a range of quality-sensitive distributed applications in P2P systems such as the *content retrieval*, and Internet-based *multimedia streaming* applications.

[2] In this paper, we will not deal with the copy right problem. We assume that the service aggregation model always chooses service components that the user is authorized to use.

3 Design and Algorithms

This section describes the design details of the *QoS-aware service aggregation* (*QSA*) model and algorithms. It enables high performance distributed application delivery in P2P systems by meeting the following challenges: (1) **Decentralization.** The solution must be fully distributed and only involve local computation based on local information; (2) **Scalability.** The solution must scale well in the presence of large number of peer nodes; (3) **Efficiency.** The solution should be able to utilize resource pools provided by P2P systems efficiently so that it can admit as many user requests as possible; and (4) **Load balance.** Although each peer makes its own decisions based on only local information, the solution should achieve the desired global properties such as load balance in P2P systems. We first state a number of key assumptions made by the service aggregation model and prove that those assumptions are valid in practice. We then describe the design details for the *on-demand service composition* and *dynamic peer selection* tiers, respectively.

3.1 Assumptions

First, we assume that application-level QoS specifications of each service instance are available and co-located with the service instance. Several programming environment and specification languages have been proposed to allow application developers to provide such QoS specifications [18, 15, 11]. Second, we assume that there exists a translator that can map the application-level QoS specifications into the resource requirements (i.e., CPU, memory, network bandwidth/latency, etc.). Such a translation procedure can be performed based on two major approaches: (1) analytical translation; and (2) offline/online probing services, which have been addressed by a wealth of research work [3, 13, 21]. Third, this paper only deals with the initial setup phase for delivering distributed applications in P2P systems, the failure detection and recovery during a session are beyond the scope of this paper and have been addressed by some research work under different context [4].

3.2 On-Demand Service Composition

We now present the *on-demand service composition* tier, which is represented by the *service composer*. In order to compose a quality consistent service path satisfying the user's end-to-end QoS requirements, the *service composer* on the user's local host needs to execute the following major protocol steps:

- **Acquire and translate the user request.** The user can directly name the requested distributed application, such as *video-on-demand*. Then the request is mapped into a list of application services, called *abstract service path*, which is created by the QoS compiler [14] or other translators. Alternatively, the user can directly define the *abstract service path* (e.g., video server → Chinese2English translator → image enhancement → video player). The user can also specify his QoS requirements using application-specific QoS parameters such as frame rate, response time.

- **Discover service instances.** Once the user request is acquired, the P2P lookup protocol, such as Chord [20] or CAN [16], is invoked to retrieve the locations (i.e., IP addresses) and QoS specifications (Q^{in}, Q^{out}, R) of all candidate service instances, according to the *abstract service path*.

- **Compose a QoS consistent shortest service path.** Due to the inherent *redundancy* property of P2P systems, multiple service instances, with different Q^{in} and Q^{out} parameters, may be returned for a required application service. Thus, the *service composer* establishes a QoS consistent shortest service path by using the information acquired in the previous step, according to the user's end-to-end QoS requirements and the resource requirements of each possible service path. We will discuss this step with more details later in this section.

- **Deliver the service path to the *dynamic peer selection* tier.** After the third step, a QoS consistent shortest service path is generated and then delivered to the *dynamic peer selection* tier.

Among the above four steps, the third one forms the key part of the *on-demand service composition* tier. It tackles the following two problems: (1) The composed service path must be QoS consistent which means that the Q^{in} of a service component must be "*satisfied*" (equ. 1, Section 2.1) by the Q^{out} of its predecessor; (2) If multiple QoS consistent service paths exist, the *service composer* should choose the one which has the minimum aggregated resource requirements so that the overall workload of a P2P system is minimized. We propose the algorithm *QCS*, the acronym of QoS consistent and shortest, to address the problems. It includes the following major operations, illustrated in Figure 3: (1) start from the (data) sink service, check the QoS consistency between the current examined service instance and all of its predecessors on the service path. If the Q^{out} of the predecessor satisfies the Q^{in} of the current examined service, then add a directed edge from the current examined service to the predecessor in the reverse direction of service aggregation flow. We assume that the Q^{out} of the sink service is set as the

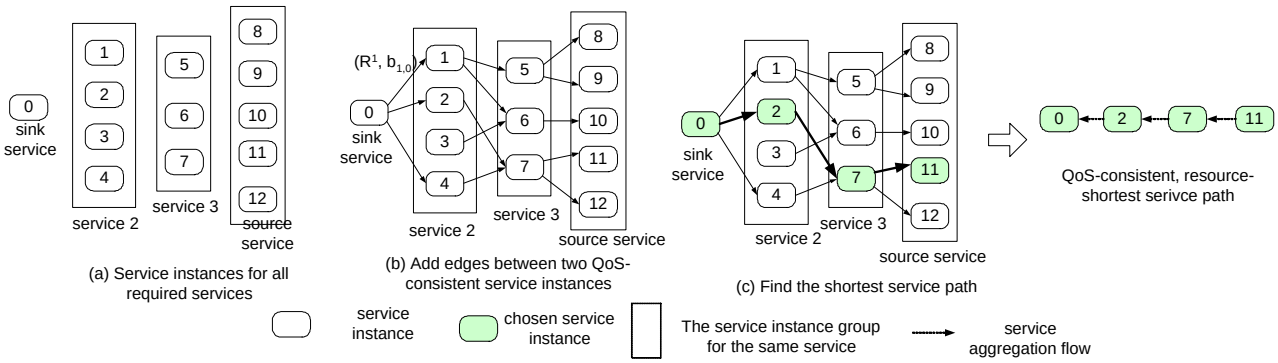

Figure 3. Illustration of on-demand service composition.

user's QoS requirements; (2) define the cost value on each edge from A to B as a resource tuple (R^B, $b_{B,A}$), where $R^B = [r_1^B, r_2^B, ..., r_m^B]$ represents the end-system resource requirements of node B and $b_{B,A}$ specifies the network bandwidth requirements from B to A [3]; and (3) find the shortest path from the data sink node to the data source node using the Dijkstra's algorithm. In order to apply the Dijkstra's algorithm, we define the comparison of any two resource tuples as follows:

DEFINITION 3.1 Given two tuples (R^B, $b_{B,A}$) and (R^D, $b_{D,C}$), they can be compared in the following way:

$$\sum_{i=1}^{m} w_i \cdot \frac{r_i^B - r_i^D}{r_i^{max}} + w_{m+1} \cdot \frac{b_{B,A} - b_{D,C}}{b^{max}} > 0$$
$$\Rightarrow (R^B, b_{B,A}) > (R^D, b_{D,C}) \qquad (2)$$

where r_i^{max}, b^{max} represent the maximum values for the ith end-system resource type and network bandwidth respectively, w_i (1 $\leq$ i $\leq$ (m+1), m is the number of all end-system resource types) are nonnegative values so that

$$\sum_{i=1}^{m+1} w_i = 1 \qquad (3)$$

For any end-system resource type r_i (e.g., CPU, disk storage), $\frac{r_i^B - r_i^D}{r_i^{max}}$ is a normalized value ranging between -1 and 1, while w_i represents its significance. Generally, we assign higher weights for more critical resources. For the network resource type, $\frac{b_{B,A} - b_{D,C}}{b^{max}}$ is a normalized value between -1 and 1, where w_{m+1} represents the importance of the network resource. For example, Figure 3 (c) illustrates such a QoS consistent shortest path (thick line). The computation complexity of the *QCS* algorithm is $O(KV^2)$, where V is the number of candidate service instances, K is the number of candidate service instances for the source service.

Thus, the final result generated by the *service composer* is a QoS consistent service path satisfying the user's end-to-end QoS requirements and also has the minimum aggregated resource requirements. However, due to the redundancy property of P2P systems, each chosen service instance can be provided by multiple different peers. Hence, we propose the *dynamic peer selection* tier to map the service instance nodes onto the peer nodes and dynamically choose the most suitable peers to execute those service instances.

3.3 Dynamic Peer Selection

This section describes the dynamic peer selection tier in the QoS-aware service aggregation (*QSA*) model. The peer selection decision is made based on the resource requirements of the service instances and the performance information of different candidate peers. However, there are two difficulties for selecting proper peers in P2P systems: (1) each peer can only maintain the up-to-date performance information for a small number of peers for scalability; and (2) the required performance information is distributed and includes multiple factors such as the peer's uptime, system load, and network bandwidth/latency. Hence, we make the following design decisions for the *dynamic peer selection* tier, represented by the *peer selector*:

- **Distributed and hop-by-hop peer selection.** Since it is impossible for each peer to have the global view of the up-to-date performance information, the peer selection has to be performed in a distributed and hop-by-hop manner. To be specific, for an n-hop service aggregation, each peer, starting from the user's host, chooses the most suitable next hop peer among all candidate peers, according to its locally maintained performance informa-

[3] Since the sink service (start point of the graph) is the common part of all possible service paths, its resource requirements are not included in the calculation.

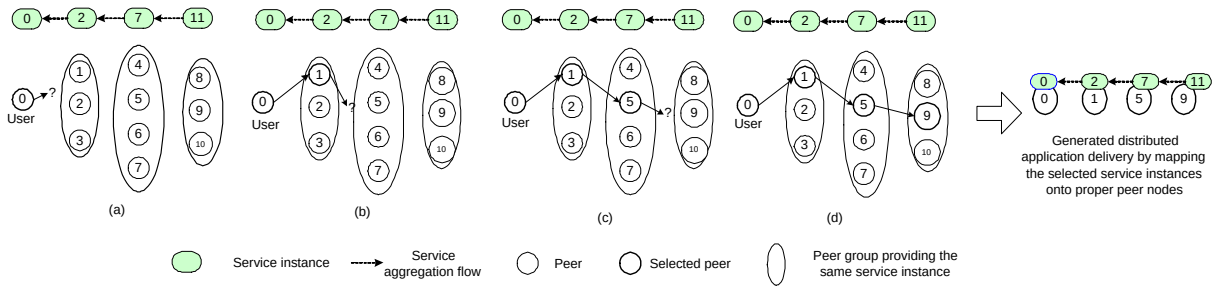

Figure 4. Illustration of distributed, hop-by-hop dynamic peer selection.

tion. Then, the peer selected in this step will be responsible for choosing the next hop peer and so on. Note that the peer selection procedure is performed in the reverse direction of the service aggregation flow. Figure 4 illustrates the process of dynamic peer selection in a four-hop service aggregation using the service path shown in Figure 3.

- **An integrated and configurable metric for peer selection.** Now we focus on a single peer selection step in which the current peer needs to choose the next hop peer according to its locally maintained performance information. If the candidate peers' performance information is not available, the peer selection falls back to a random policy. Otherwise, the *peer selector* first chooses among all candidate peers according to the match between (1) the candidate peer's uptime and the application's session duration for better tolerance to P2P systems' topological variation [4]; (2) the candidate peer's resource availability and the service instance's resource requirements. Second, if multiple peers qualifies, the *peer selector* uses an integrated and configurable metric Φ to choose the best one. The metric Φ is proposed to tackle the problem of composite-value decision-making for peer selection. For this purpose, we define RA as the candidate peer's end-system resource availability, and β as the end-to-end available network bandwidth from the candidate peer to the current peer. The resource availability vector RA and the resource requirement vector R represent the same set of resources and obey the same order. In addition, we define b as the service instance's required network bandwidth. Based on the above definitions, the metric Φ can be defined as follows:

$$\Phi = \sum_{i=1}^{m} \omega_i \cdot \frac{ra_i}{r_i} + \omega_{m+1} \cdot \frac{\beta}{b} \quad (4)$$

where ω_i ($1 \leq$ i $\leq$ (m+1)) are nonnegative values so that

$$\sum_{i=1}^{m+1} \omega_i = 1 \quad (5)$$

The *peer selector* chooses the best candidate peer which maximizes the value of the above metric Φ. The value of Φ represents the *advantage* of selecting a specific peer to execute the service instance. First, for any end-system resource type r_i (e.g., CPU, memory, disk storage), $\frac{ra_i}{r_i}$ is a normalized value which means that the larger ratio between the resource availability and the resource requirement is, the more advantageous it is to select this peer for achieving load balance in heterogeneous P2P systems. $\frac{\beta}{b}$ represents the same meaning for the network bandwidth. In order to allow customization, we introduce ω_i $(1 \leq i \leq m+1)$ to represent the importance of the ith resource type in making the peer selection decision. They can be adaptively configured according the application's semantics and user's preference.

- **Dynamic neighbor resolution.** According to the neighbor definition introduced in Section 2.2, the resolution of the neighbor list at each peer is dynamic and depends on the results from the *service composer*. Thus, the neighbor list at each peer is updated using the *dynamic neighbor resolution protocol*. To be specific, after the *service composer* generates a service path, it first updates the local host's direct neighbor list to include those candidate peers which provide the services on the service path. Then it notifies all candidate peers to update their indirect neighbor list to include those peers which provide the preceding services on the service path and so on. The neighbor list at each peer is maintained as soft states. Thus, the above notification messages are sent periodically to refresh the soft states as long as the service path is valid and needed.

[4]The uptimes of all participating peers must be greater than the application's session duration. Otherwise, if any of them leaves during the session, the application delivery fails.

At the end of the *dynamic peer selection* phase, the distributed application delivery can be started by backtracking the generated *peer path*.

4 Simulation Results

In this section, we evaluate the performance of the QoS-aware service aggregation (*QSA*) model by simulation. We first describe our evaluation methodology. Then we present and analyze the simulation results.

4.1 Evaluation Methodology

We simulated a large-scale P2P system of 10^4 peers. Each peer is randomly assigned an initial resource availability $RA = [cpu, memory]$, ranging from [100,100] to [1000,1000] units. Different units reflect the heterogeneity in P2P systems. For example, if we assign [100,100] units to a laptop, then we probably need to assign [500, 500] units to a desktop PC, and [1000, 1000] to a powerful cluster-based server. The end-to-end available network bandwidth between any two peers is defined as the bottleneck bandwidth along the network path between two peers, which is initialized randomly as 10M, 500k, 100k, or 56k bps. The network latency between two peers are also randomly set as 200, 150, 80, 20, or 1 ms [12].

The *QSA* algorithms are locally executed on each peer. They include processing user request, composing services, selecting peers and periodically probing neighbor peers. In our experiments, the maximum number of neighbor peers any peer can probe (M) is 100 so as to control the probing overhead within $100/10000 = 1\%$. During each minute, certain number of user requests are generated and assigned on a set of randomly chosen peers. The user request is represented by any of the 10 distributed applications whose service path lengths are between 2 to 5 and whose session durations are between 1 to 60 minutes. The user's QoS requirement is specified by a single parameter which has three levels: high, average, and low. Each service instance is also randomly assigned values for its Q^{in}, Q^{out} and R parameters. The number of different service instances for each service is randomly set between 10 to 20. The number of peers which provide a specific service instance is randomly set between 40 to 80. The importance weights for different resource types are uniformly distributed.

The metric we use for evaluating the performance of the *QSA* algorithm is the *service aggregation request success ratio*, called ψ. A service aggregation request is said to be successful if and only if during the entire application session, all service instances' resource requirements are always satisfied by the resource availability along the aggregation path. On other words, a service aggregation request is failed when its resource requirements cannot be satisfied or one of provisioning peers leaves during the session. The metric ψ is defined as the number of successful requests over the total number of all requests. Higher service aggregation request success ratio represents improved service availability and better load balance in heterogenous P2P systems. For comparison, we also implement two common heuristic algorithms: *random* and *fixed* algorithms. The *random* algorithm randomly chooses a QoS consistent service path (without considering the aggregated resource consumption) and randomly selects a set of provisioning peers for instantiating the service path. The *fixed* algorithm always picks the same service path for a distributed application delivery and chooses the dedicated peers to instantiate the service path. The *fixed* algorithm actually represents the conventional client-server systems.

The major goals of the *QSA* algorithm are to achieve better performance and higher tolerance to P2P systems' topological variation. Hence, we conduct two sets of experiments to evaluate how well the *QSA* algorithm meets these goals. In the first set of experiments, we assume that there is no topological variation in P2P systems. We study the *QSA*'s ability to achieve better performance than the *random* and *fixed* algorithms. In the second set of experiments, we consider the topological variation in P2P systems. We study the *QSA*'s resilience to the topological variation, compared with the *random* and *fixed* algorithms. In both sets of experiments, we use the *service aggregation request success ratio* ψ as the performance metric.

4.2 Results and Analysis

Figure 5 and Figure 6 shows the simulation results for the first set of experiments which do not consider the topological variation in P2P systems. In Figure 5, the X axis represents different service aggregation request rate, calculated by the number of requests per minute. The range of request rate is selected to reflect different workload of the P2P system. The Y axis shows the average service aggregation success ratio (ψ) achieved by the *QSA*, *random* and *fixed* algorithms. Each average success ratio value is calculated and averaged over a period of 400 minutes. The results show that the average success ratio of the *QSA* algorithm is always higher than the other two heuristic algorithms under all request rates. The reason is that the *QSA* algorithm reduces the overall workload of the P2P system by choosing the service path with minimum aggregated resource consumption. Moreover, the *QSA* algorithm achieves better load balance by always selecting the peers which have the most abundant resources. The *random* algorithm achieves lower success ratios than the *QSA* algorithm,

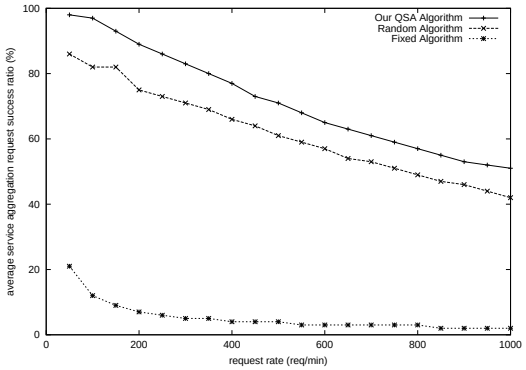

Figure 5. Average success ratio under different service aggregation request rates , over a period of 400 minutes without topological variation.

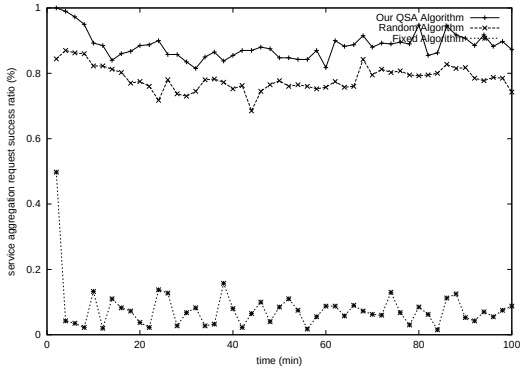

Figure 6. Success ratio fluctuation within a period of 100 minutes, for the request rate = 200 req/min without topological variation.

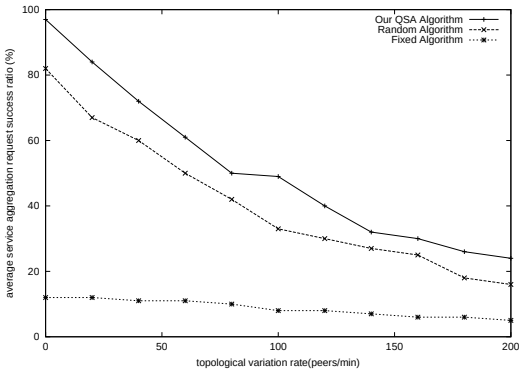

Figure 7. Average success ratio under different topological variation rate (peers/min) over a period of 60 minutes with request rate = 100 req/min.

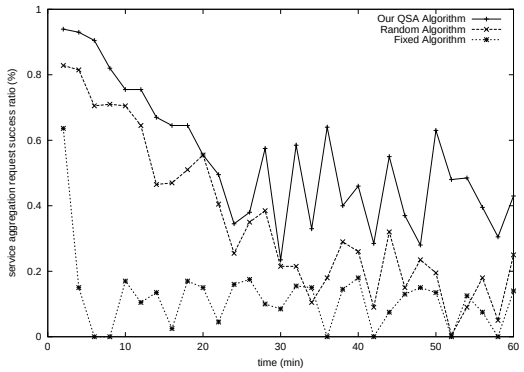

Figure 8. Success ratio fluctuation within a period of 60 minutes, for the request rate = 100 req/min, and topological variation rate = 100 peers/min.

but much higher success ratios than the *fixed* algorithm. Such results reflect the prominent advantage of P2P systems compared to the traditional client-server systems due to its natural redundancy property for better load balance. Figure 6 gives a more detailed picture about the success ratio fluctuations under a particular request rate (200 requests/minute). Each run of simulation lasts 100 minutes and the success ratio value is sampled every 2 minutes. We observe that the success ratio of *QSA* is consistently higher than those of *random* and *fixed*. The former may be higher than the latter two as much as 15% and 90%, respectively.

The second set of simulation results are illustrated in Figure 7 and Figure 8. In this set of experiments, we consider the topological variation in P2P systems, which is measured by the number of peers leaving or arriving every minute. Figure 7 shows the average success ratios achieved by *QSA*, *random* and *fixed* under different topological variation rates. Each run of simulation lasts 60 minutes under a fixed request rate (100 requests/minute). Figure 8 shows the success ratio fluctuations for a fixed topological variation rate (100 peers/minute) with a particular request rate (100 requests/minute). Both simulation results prove that *QSA* tolerates topological variation best and uniformly achieves the highest success ratio. The reason is that when *QSA* selects among candidate peers for instantiating a service instance, it takes the peers' average uptimes into account but *random* and *fixed* do not. The *QSA* algorithm always chooses those peers connected to the P2P system for an average duration which is longer than the application's session duration in hope that those peers will continue to be con-

nected with the P2P system for at least the same duration. However, such a heuristic cannot be true all the time. Hence, the results in Figure 7 and Figure 8 show that the performance of P2P systems is very sensitive to the topological variation, even with a small number of peer arrivals/departures ($\leq$ 2% total peers). Under such circumstances, we do need runtime failure detection and recovery to improve the performance.

5 Related Work

Recently, P2P systems have drawn much research attention with the popularity of P2P file sharing applications such as Gnutella and Napster. However, most of the research work, such as Chord [20] and CAN [16], focuses on providing an efficient lookup service which enables the data (e.g., mp3 files) and resource (e.g., disk storage) sharing in P2P systems. We believe that in order to make P2P systems become more successful, a scalable and QoS-aware *service sharing* support is also necessary so that various high performance distributed applications such as *media-on-demand* and *value-added information processing* can be accessed by the users of P2P systems. Hence, we remedy this situation by proposing a scalable, QoS-aware service aggregation model for P2P systems.

The problem of service aggregation has been addressed by different research work. In the Ninja project [9], service aggregation is performed by composing a sequence of application-level service operators and connectors into a customized delivery to heterogeneous clients, which is called a *service path*. Since the Ninja project aims to support client-server based Internet services, it presents several limitations when applying to peer-to-peer systems. For example, it does not support dynamic peer selection since all services are assumed to be provided by dedicated servers. Also it does not consider the dynamic and ad hoc membership problem presented in P2P systems. The authors in [22] proposed an application-level solution for finding multimedia service path in the media service proxy network. The service path finding takes into account the end-to-end resource availability and is integrated with a resource monitoring mechanism. However, their solution requires each media proxy to maintain a global view of the entire media proxy network in terms of resource availability. Hence, their solution only applies to the small size proxy network, and thus does not meet the scalability requirement of P2P systems. Finally, in [10], we proposed a centralized approach to address the service composition and instantiation problems in ubiquitous computing environments. We achieved scalability by applying a hierarchical design to the ubiquitous computing environment. However, such a solution is not suitable for P2P systems which have neither centralized infrastructure support nor hierarchical organization. Hence, the originality of this paper comes from the fact that we target to large-scale P2P systems and meet the scalability challenge by using a fully distributed solution.

Other closely related work includes different server selection solutions for the Internet applications such as the World Wide Web. In [6], the authors propose a dynamic server selection scheme which is based on the online measurements of network bandwidth and latency. Also in [19], the authors study the effectiveness of DNS-based sever selection scheme which redirects the client requests to the closest point of service. Our dynamic peer selection algorithm shares the same idea with the above work that application-level services should be instantiated selectively based on the dynamic performance information such as network bandwidth, latency and distance (hop count). However, since the server selection solutions only deal with a fixed set of dedicated servers, they did not meet the new challenges in P2P systems which are addressed by our dynamic peer selection solution. For instance, peer selection must consider the peer's uptime to deal with the dynamic membership problem in P2P systems, for assuring high service availability. Moreover, the dynamic peer selection must be performed in a distributed manner in order to meet the scalability requirements of P2P systems.

6 Conclusion and Future Work

We have presented a scalable QoS-aware service aggregation model for providing high performance distributed applications in peer-to-peer systems. The major contributions of the paper are as follows: (1) solve two key problems, *service composition* and *peer selection* in an integrated service aggregation model; (2) present an *on-demand service composition* algorithm which generates a quality consistent service path with minimum aggregated resource requirements while satisfying the user's end-to-end QoS requirements; (3) provide a scalable dynamic peer selection scheme for instantiating the service path. We have implemented a simulation testbed and our initial simulation results illustrate the effectiveness of the proposed model and algorithms. In the future, we will implement a prototype of our model and test it in the real Internet environment. We will also investigate how to include other desirable system properties such as stability and fault tolerance into our model.

7 Acknowledgment

This work was supported by the NASA grant under contract number NASA NAG 2-1406, NSF under contract number 9870736, 9970139, and EIA 99-72884EQ.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NASA, NSF or U.S. Government. We would like to thank anonymous reviewers for their helpful comments.

References

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

[2] Napster. *http://www.napster.com/*.

[3] Tarek F. Abdelzaher. An Antomated Profiling Subsystem for QoS-Aware Services. *Proc. of IEEE Real-Time Technology and Applications Symposium*, June 2000.

[4] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. *Proc. 18th ACM SOSP 2001, Banff, Canada*, October 2001.

[5] A. P. Black, J. Huang, and J. Walpole. Reifying Communication at the Application Level. *Proc. of ACM Multimedia (Multimedia Middleware Workshop)*, October 2001.

[6] Robert L. Carter and M. E. Crovella. Server Selection using Dynamic Path Characterization in Wide-Area Networks. *Proc. of IEEE Infocom 1997, Kobe, Japan*, April 1997.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[8] P. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The Architecture of the Remos System. *Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing(HPDC 2001)*, August 2001.

[9] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Computer Networks, Special Issue on Pervasive Computing*, 2001.

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

[11] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. *Journal of Visual Language and Computing, Special Issue on Multimedia Language for the Web*, 2002.

[12] Kevin Lai and Mary Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, California*, March 2001.

[13] B. Li and K. Nahrstedt. QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications. *Proc. of IFIP International Conference on Distrbited Systems Platforms and Open Distributed Processing (Middleware 2000)*, April 2000.

[14] K. Nahrstedt, D. Wichadakul, and D. Xu. Distributed QoS Compilation and Runtime Instantiation. *Proc. of IEEE/IFIP International Workshop on QoS 2000(IWQoS2000)*, June 2000.

[15] Joseph P.Loyall, David E. Bakken, Richard E.Schantz, John A.Zinky, David A.karr, Rodrigo Vanegas, and Kenneth R.Anderson. QoS Aspect Languages and Their Runtime Integration. *Lecture Notes in Computer Science*, 1511, May 1998.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *Proc. of the ACM SIGCOMM 2001, San Diego, CA*, 2001.

[17] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proc. of SPIE Multimedia Computing and Networking 2002 (MMCN'02),San Jose, California*, 2002.

[18] S.Frolund and J.Koistinen. QML: A Language for Quality of Service Specification. *Technical Report HPL-98-10,*, February 1998.

[19] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. *Proc. of IEEE Infocom 2001, Anchorage, Alaska*, April 2001.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proc. ACM SIGCOMM 2001, San Diego, California*, 2001.

[21] D. Wichadakul, K. Nahrstedt, X. Gu, and D. Xu. 2KQ+: An Integrated Approach of QoS Compilation and Component-Based, Runtime Middleware for the Unified QoS Management Framework. *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[22] D. Xu and K. Nahrstedt. Finding Service Paths in a Media Service Proxy Network. *Proc. of SPIE/ACM Multimedia Computing and Networking Conference (MMCN'02), San Jose, CA*, January 2002.