Coordinated Service Provision in Peer-to-Peer Environments

Gang Chen, Chor Ping Low, *Member*, *IEEE Computer Society*, and Zhonghua Yang, *Senior Member*, *IEEE*

Abstract—In recent years, inspired by the emerging Web services standard and peer-to-peer technology, a new federated service providing (FSP) system paradigm has attracted increasing research interests. Many existing systems have either explicitly or implicitly followed this paradigm. Instead of exchanging files, peers in FSP systems share their computation resources in order to offer domain-specific services. In this paper, we focused on the coordination problem of how to self-organize the service group structures in response to the varying service demand. We presented our solution in the form of a coordination mechanism, which includes a labor-market model, a recruiting protocol, and a policy-driven decision architecture. Peers make their service providing decisions based on their local policies, which can be added, removed, or modified by users. A general methodology is introduced in this paper to facilitate policy design. Specifically, a heuristic inspired by the Extremal Optimization technique is utilized to handle potential inconsistencies among policies. A stimulus-response mechanism was further applied to make the decision process adjustable. Experiments under five application scenarios verified our ideas and demonstrated the effectiveness of our coordination mechanism.

Index Terms—Federated service providing system, P2P system, coordination, decision policy.

1 Introduction

The booming Internet industry has brought many distributed systems that were dreams yesterday into reality [4], [7], [16]. Among them, the peer-to-peer (P2P) file sharing system now wins an astounding popularity and draws much research attention [1], [2]. P2P systems resemble a cost-efficient way of aggregating a tremendous amount of resources, and there lies their major advantage. Previous research efforts focus mainly on the problems of scalable data lookup and effective data sharing in P2P systems [27], [32], [35]. Recently, inspired by the emerging Web service standards and middleware technology, a new *federated service providing* (FSP) system paradigm is formed, where Internet applications will be comprised of multiple service components distributed across numerous network nodes (that is, peers) in collaborative virtual organizations [15], [20], [21]. The FSP system concept has been widely explored (either explicitly or implicitly) in many distributed applications such as multimedia processing [20], data mining [34], bioinformatics [12], and Universal Description Discovery and Integration (UDDI) [15]. One specific application that has inspired the research reported in this paper is to construct a *self-adaptive* semantic Web service discovery system [6], [33], as illustrated in Fig. 1.

Similar to those in the work of Cuenca-Acuna and Nguyen [15], peers as shown in Fig. 1 provide Web *service discovery* services. Service discovery is achieved by *matching* service requests with the *ontology-based service descriptions* of registered Web services [25]. According to the essential ontologies involved, it is convenient to differentiate several types of services, such as s^1 to s^4 in Fig. 1. Depending on the type of services offered, a peer may belong to varied service groups. Every user of the FSP system is able to access services provided within any service groups. Comparing with a centralized architecture, the FSP system shown in Fig. 1 is more scalable because 1) the logic inference involved in matching service requests is dispersed across the peers in different service groups, and 2) the complexity of the logic inference is further reduced as peers rely primarily on ontologies of particular domains.

Intuitively, when the demand for a certain service such as s^3 changes, the service group G^3 corresponding to s^3 should change accordingly. An FSP system is considered *self-adaptive* if it can dynamically adjust its service group structures so as to reflect the changing service demand and improve users' satisfaction. Such self-adaptability is desirable as it relieves a considerable administrative burden from human operators. To make any FSP system such as the one in Fig. 1 *self-adaptive*, we must create effective coordination mechanisms to manage the services provided by peers. Few people have addressed this problem in the literature. This motivates our research, and a study of such a coordination mechanism is provided in this paper. Several research contributions in the process of developing our coordination mechanism have been made in this paper. They are summarized as follows:

1. A *labor-market model* is proposed by us to form the basis of a *recruiting protocol* that controls the interaction between peers that belong to the same or different service groups.
2. Our solution describes a policy-based decision-making architecture toward achieving coordinated

• *The authors are with the Information Communication Institute of Singapore, School of Electrical and Electronics Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: {chengang, icplow, ezhyang}@ntu.edu.sg.*

Manuscript received 26 Oct. 2006; revised 17 Apr. 2007; accepted 10 July 2007; published online 24 July 2007.
Recommended for acceptance by J. Hou.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0341-1006.
Digital Object Identifier no. 10.1109/TPDS.2007.70745.

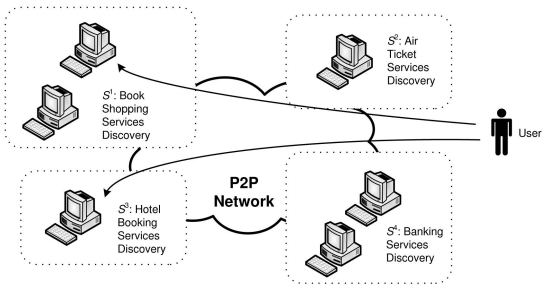

Fig. 1. A semantic Web service discovery system.

service provision among peers. A peer's decision at various stages of a recruiting process is guided by its local policies. A formal methodology for policy design has been proposed and explored in this paper.

3. Policies involved in making a certain decision may contradict with each other to a certain extent. To solve this problem, a conflict resolution mechanism adopted from a technique termed *Extremal Optimization* (EO) [8] has been introduced in this paper. A *stimulus-response mechanism* is further utilized to make the decision process more adjustable. According to our knowledge, the combination of EO and the stimulus-response mechanism improves the decision flexibility and is a unique feature of our policy-based decision-making architecture.

Experiments have been conducted to verify all of our ideas in a simulated FSP system. Our experiment results indicate that the proposed coordination mechanism is effective as the average response time to service requests is well balanced and kept small. The experiments under different system configurations also reveal that our coordination mechanism is scalable and quite robust with respect to service requirement changes.

The remainder of this paper is organized as follows: Section 2 summarizes and compares related research. Section 3 describes the FSP system, as well as basic system assumptions. In Section 4, the labor-market model is introduced together with the recruiting protocol. Section 5 covers the policy-based decision architecture and policy design methodology. The effectiveness of our coordination mechanism is assessed in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

The idea of utilizing the local resources (for example, CPU cycles) of multiple machines in a distributed environment has been greatly advanced by the recent research progress in Grid computing [7]. Traditional Grid systems are often based on variants of the master-worker paradigm [24], [31]. In heterogeneous environments, very different application execution times can be witnessed depending on the choice of both masters and workers [31]. At another extreme of Grid systems is the so-called desktop grid [26]. In desktop grid, desktops typically interact with each other directly without being controlled by masters. Chakravarti et al. further proposed an organic grid system that enables desktops to self-organize their task executions [12]. Desktops in the grid environment share their resources and together finish a given set of tasks. In comparison, in the FSP system, each service request from end users will be processed by a single peer only.

FSP systems have received increasing research interests recently. Various research issues with sufficient merit in specific application domains have been studied. For example, Gu and Nahrstedt considered a service composition problem in a P2P streaming environment [20]. They constructed a VoIP application where a speaker's audio stream is to be processed by various language processing services provided by peers. The problem is to find a proper processing workflow that links multiple peers together. Ratsimor et al. have studied the service discovery problem in a mobile P2P environment [28]. They developed a framework called *Allia* to facilitate service caching and discovery. Due to the use of a policy-driven system architecture, their approach can adapt to changes in the environment with a high degree of flexibility. Different from these research works, our research focuses on a common coordination problem. Services in this paper are viewed as general entities without considering their actual functionalities.

The need to automate service deployment and management has been identified by Cuenca-Acuna and Nguyen [15]. They considered the UDDI service, which is typically replicated across multiple peers. They proposed a distributed resource management framework with the aim of controlling the set of peers (termed a configuration) that can host instances of a single service. An optimization objective is defined for the UDDI service and the genetic algorithm (GA) [18] is utilized periodically to find new configurations based on the system status update. Although Cuenca-Acuna and Nguyen have addressed a problem like ours, there are several differences between the two research works:

1. Cuenca-Acuna and Nguyen [15] focused on managing a single service, whereas we considered the problem of managing several types of services,
2. GA is used in [15] to find service configurations based on *global system information*, whereas peers in our system rely mainly on their local policies and do not use any optimization algorithms, and
3. the peer running a deployment planning component in [15] needs to contact every other peer in the system, whereas every peer in this paper only needs to contact a few selected peers.

In Section 6.2, we will further compare Cuenca-Acuna and Nguyen's framework with our coordination mechanism through experiments.

There has been much research into mediating the interaction among distributed entities through the market-system paradigm [3], [11], [22], [23]. For example, Hausheer and Stiller developed a system called *PeerMart* to support a decentralized market environment for trading P2P services [22]. Different from this and similar approaches, in our labor-market model, peers interact directly with other peers in different service groups without relying on any broker peers. The goal is not to obtain a service but to change the current service provided by these peers. No concepts such as utilities or prices are involved.

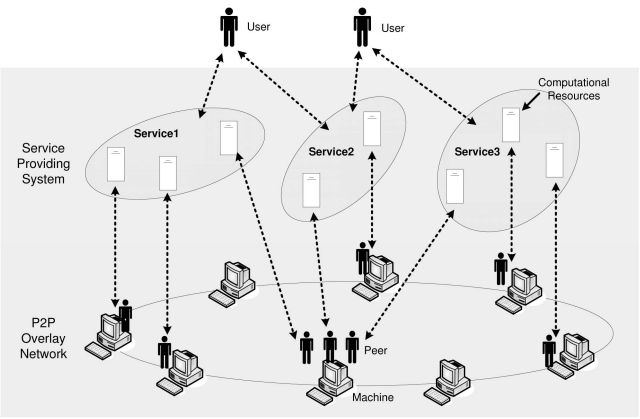

Fig. 2. The FSP system architecture.

3 The Federated Service Providing System

In an *FSP system*, each service is viewed as a general unit of functionality. A peer that provides a service is termed an *instance* of the service. A user who wants to access a service issues a service request, which will be handled subsequently by a service instance. At the conceptual level, the FSP system model is illustrated in Fig. 2.

As shown in Fig. 2, the FSP system architecture is comprised of two layers: the *overlay network layer* (ONL) and the *service providing layer* (SPL). At the bottom of the system architecture is the P2P ONL, which provides the basic network facilities to support peer lookup and communication. When a peer receives a service request that cannot be processed by itself, either because it is busy or because the requested service type is different, the peer needs to find other peers capable of processing this request. For this reason, ONL should be able to locate peers that provide any type of services. One solution to satisfy this requirement is to follow the decentralized service management framework proposed by Gu et al. based on the distributed hash table (DHT) system [21], [30].

Simply put, when a peer p^1 wants to provide a service s^1, it registers itself into the DHT system with a key that uniquely identifies the type of service s^1 it provides. As those peers that offer the same type of service share the same key, the DHT system will register them on the same DHT-assigned peer, for example, p^2. In the case that another peer p^3 wants to locate peers offering service s^1, p^3 can generate a query message using the key associated with service s^1. This message will be routed to the assigned peer p^2 by the DHT system. p^3 is then able to discover from p^2 that p^1 actually provides service s^1. It is to be noted that instead of the DHT technology, unstructured P2P networks have managed to occupy a significant portion of today's Internet bandwidth [1], [2]. The great success of unstructured P2P networks such as BitTorrent suggests that directory servers may be utilized by ONL in order to keep track of peers in every service group. In fact, our FSP system does not restrict the type of P2P overlay networks to be adopted in ONL. As ONL is not the focus of this paper, we will not continue on this topic anymore.

SPL manages the actual service provision and is the focus of this paper. Every peer in the FSP system runs a piece of code to implement the SPL functionality. In SPL, peers are assumed to be *cooperative* in nature. Multiple peers can also reside in the same network machine [17]. A peer therefore simply represents a certain amount of resources shared by a machine. Although a peer is only allowed to provide a single type of service at any time, a network machine is capable of offering several types of services concurrently. For example, in Fig. 2, three types of services are provided by the FSP system. The *instances* of each service are grouped together to form a separate *service group*. By hosting one instance in every service group, a machine is able to offer all the three services. As "*peer*" and "*machine*" are not identical in concept, certain peers may become unavailable temporarily when the total amount of resources shared by all peers hosted in the same machine decreases (for example, due to competition among peers for scarce resources). We will go back to this topic in Section 6.6.

4 The Labor-Market Model and the Recruiting Protocol

As mentioned in Section 1, peers in an FSP system will form service groups depending on the types of services they provide. To further regulate the interactions within and between service groups, a *labor-market model* is proposed by us as the basis of a *recruiting protocol*, which controls the services provided by peers.

4.1 The Labor-Market Model

The labor-market model establishes an *analogy* between FSP systems and a simplified social recruiting structure. In the literature, sociology structures have long served as general heuristics to solve problems from trust forming [36], resource discovery and allocation [37], to decentralized scheduling [29] in distributed computing environments. The labor-market model described in this paper aims at managing the service group structures so as to 1) achieve system performance requirements and 2) adapt to varying service demands through self-organization. Several modeling concepts as detailed below are essential to the understanding of the labor-market model:

- **Peer.** This is denoted by p^i, where i refers to the universal peer ID (PID). A set of peers is represented as $\{p^i\}$. A peer has several important properties such as the *service* it is providing and the *computation resources* it is willing to contribute to provide this service. Normally, computation resources are measured in terms of a peer's *processing capability*, which is defined as the number of computations (CPU cycles) allowed by the peer per second. A peer is capable of processing service requests one at a time. At any time t, the request being processed by peer p^i is denoted by $SrvReq(p^i, t)$.
- **Service.** We use s^j to represent a service, where j refers to the universal service ID (SID). $\{s^i\}$ stands for a set of services. Service is a self-contained unit of domain-specific functionality. "*Self-contained*" means that providing one service does not depend on the availability of other services. Service is to be provided by peers who are termed *service instances*.
- **Service request.** Users are allowed to submit any number of service requests to the FSP system. A

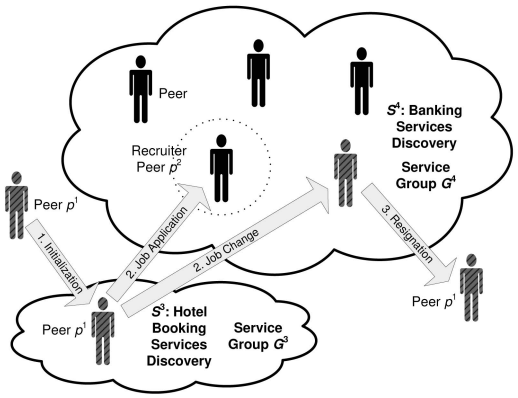

Fig. 3. A simple scenario of the labor-market model.

Precondition:

1) Peer p^i belongs to a service group $G =< \{p^i\}, s >$.
2) Peer p^i is not handling any service requests, $SrvReq(p^i, t) = \emptyset$.
3) Peer p^i decides to resign based on its **Resignation Decision** that involves the local information of other $m-1$ peers in G.

Action:

1) p^i removes its group membership from G.
2) p^i changes its group belonging status to $p^i \in \emptyset$.
3) p^i forwards all incoming requests for service s to peers in group G.

Fig. 4. The resignation protocol.

request for service s^j submitted at time t is denoted as $Re_t^{s^j}$. $Re_t^{s^j}$ may finally be processed by a peer p^i, which is an instance of s^j. The processing of $Re_t^{s^j}$ will consume a certain amount of computation resources of p^i. A service request is completed after the user receives a service reply for his request.

- **Service group.** A service group is comprised of a set of peers $\{p^i\}$ dedicated to providing a single service s^j and is represented as a two-element tuple $G =< \{p^i\}, s^j >$. The size of G is identical to the cardinality of set $\{p^i\}$, $\|\{p^i\}\|$. We say that a peer p^i is recruited by group G if and only if $p^i \in \{p^i\}$. A peer can be recruited by at most one service group at any time. However, it can choose not to work for any service groups. In this case, the peer becomes *unemployment*, namely, $p^i \in \emptyset$. $\emptyset$ stands for the *idle group*. The shared resources of those peers that belong to the idle group may be utilized for other purposes but are not considered in this paper. Using our definition of service groups, the *global group structure* is further defined as a set of groups $GS = \{G^1, G^2, \cdots, G^n\}$. For any service s^j offered by the FSP system, there is exactly one service group G^j in GS that provides service s^j.

In addition to the above four concepts, we have further identified several key *events* involved in altering peers' group membership. They together form the basis of a *recruiting process*:

- **Resignation.** At a certain point of time t, a peer p^i may decide to quit its current job in a service group G. This event changes p^i's group membership from G to $\emptyset$, $p^i \in G \xrightarrow{t} p^i \in \emptyset$.
- **Job Change.** Any peer p^i can apply for joining a service group (for example, G^1) that is different from its current group (for example, G^2). This application is called a *job application*. Once the application is *approved*, the job-change event will alter p^i's group membership from G^2 to G^1, $p^i \in G^2 \xrightarrow{t} p^i \in G^1$.
- **Initialization.** When a peer p^i becomes available, its initial group membership is determined after an initialization event.

To help demonstrate the recruiting process, which is a combination of the three key events above, Fig. 3 highlights the potential activities of one selected peer in the FSP system. As shown in Fig. 3, a peer p^1 is initially assigned to provide service s^3. After a while, p^1 intends to change its job and provide service s^4 instead. It sends a *job application* to a randomly selected peer p^2 that actually provides service s^4. Every peer in a service group has the responsibility of handling incoming job applications and therefore can serve as a *recruiter*. After p^1's job application has been approved by p^2, p^1 triggers a job-change event and starts to provide service s^4. Finally, a resignation event happens, and p^1 becomes unemployment.

Our simple scenario reveals two qualitative properties of the labor-market model. First, peers are encouraged to stay in their respective service groups as changing jobs will incur extra communication and processing costs. This is in accordance with our objective to maintain the stability of the group structure. Second, the labor-market model provides enough flexibility to adapt to the varied service demand. When a certain service has a high demand, other peers may choose to join the corresponding service group despite of the joining cost. Peers can also choose to resign when the service demand drops in order to save their resources.

4.2 The Recruiting Protocol

In order to exploit the labor-market model in practice, the recruiting process is interpreted in terms of a distributed *recruiting protocol* in this paper. We break the recruiting protocol into three *subprotocols* and name them, respectively, as the *resignation protocol*, the *job-application protocol*, and the *employment protocol*. The resignation protocol implements the resignation event. As at least one *job-applicant peer* and one *recruiter peer* are engaged in a job-change event, the activities of these two peers are described separately in two protocols. The job-application protocol controls the activities of the job-applicant peer, whereas the employment protocol regulates the activities of the recruiter peer. Figs. 4, 5, and 6 show the workflow of the three subprotocols. Each subprotocol is comprised of two parts, namely, the precondition part and the action part. When the preconditions are satisfied, the corresponding actions will be performed accordingly. A peer will periodically check the satisfaction of all the preconditions.

Precondition:
1) Peer p^i receives a job application from peer p^j.
2) Peer p^i belongs to a service group G.

Action:
1) p^i collects the local information of other $m-1$ peers in group G.
2) p^i uses its collected information to derive an **Employment Decision** as whether or not to employ p^j.
   a) **IF** the decision is to employ, p^i notifies p^j with an approval message.
   b) **ELSE** p^i notifies p^j with a rejection message.

Fig. 5. The employment protocol.

We believe that the workflows in Figs. 4, 5, and 6 are just elaborations of our previous discussions and are self-explanatory. For this reason, we will not detail them further. The local information of any peer p^i involved in the three subprotocols refer primarily to the range of information that will affect the service provision process of p^i. More details can be found in Section 5.2.

It is to be noted that no protocols are defined to cover the initialization event. This is because the group structure at a later stage, in principle, should not be significantly influenced by the initial assignment of peers to service groups due to group self-organization. In our simulation system, all peers are randomly assigned to a service group with equal probability when they start operating. This requirement is very easy to achieve in a P2P environment. Please refer to Section 6 for more information.

The recruiting protocol contains two tunable parameters, that is, the m peers to contact in the resignation and employment protocol and the h peers to send job applications

Precondition:
1) If peer p^i belongs to a service group $G =< \{p^i\}, s >$, then p^i must have already served in G for certain period of time determined by users.

Action:
1) p^i randomly selects h service groups.
2) p^i randomly selects one peer p^j from each of the chosen service group.
3) p^i sends a job application to every of the h peers (i.e. p^j).
4) p^i receives the reply from the h peers.
   a) **IF** at least one peer approved p^i's job application.
      i) p^i decides to join a service group G' according to its **Group-Selection Decision**.
      ii) p^i continues to process its current service request, after which it will only accept service request for G'.
      iii) p^i updates its group membership in G' and removes itself from G.
   b) **ELSE** p^i continues its job in group G

Fig. 6. The job-application protocol.

1) Identify the decision to be made by a policy.
2) Identify the decision-related criterion Cr to be utilized by the policy under design.
3) Construct a method to evaluate Cr by using a peer's local information.
4) Establish a *total order relation* $\leq_{Cr}$ among all peers involved in the decision-making.
5) Give each peer a rank r according to the ordered relation and identify a decision threshold θ derived from the ranks assigned to peers.
6) Specify a method to derive the decision result based on the threshold θ.

Fig. 7. The major steps involved in constructing a decision policy.

in the job-application protocol. By enlarging m, the peers' local decisions will have a better chance to benefit the service groups they belong to. Likewise, by enlarging h, a peer might be able to find more job opportunities at the price of an increased employment cost. The choice of m and h depends on the specific applications. As a simple heuristic, if the number of peers in the FSP system is upper bounded by N_p and the number of service groups is upper bounded by N_g, we might choose $m = N_p/N_g$ and $h = N_g/2$. This is exactly the heuristic used in our simulation system.

5 Policy-Based Decision Heuristics

The recruiting protocol requires a peer to make three different types of decisions, which are to

1. decide whether to resign (Fig. 4),
2. decide whether to employ (Fig. 5), and
3. decide which service group to join (Fig. 6).

A policy-driven architecture is introduced in this paper to control the decision-making process. We view a *policy* as a rule that governs the condition of making certain decisions. A policy is said to be *inapplicable* if the conditions stipulated by the policy are not satisfied. Policies may potentially reflect a combination of users' preferences, performance considerations, and technical feasibility. A peer's local policies are managed through its *policy manager*. Whenever a peer needs to make certain decisions, it must consult its policy manager. This ensures that the peer's decisions will always comply with its local policies.

We have developed a methodology to facilitate policy design in this paper. We first list the major steps involved in designing a policy and then explain these steps through an example (that is, design the resignation policies for our experiments in Section 6). The six policy design steps are shown in Fig. 7. The first step is to clarify the purpose of a policy $\mathbf{P}$ under design. For example, if policy $\mathbf{P}$ controls the resignation decision, it is termed a *resignation policy*. Likewise, there are the *employment policy* and the *group-selection policy* as well. Suppose policy $\mathbf{P}$ is a resignation policy of peer p^i. If $\mathbf{P}$ is inapplicable, peer p^i will not decide to resign. In step 4 in Fig. 7, the peers involved in making a decision refer to 1) the m peers in the resignation or

employment decision or 2) the h peers in the group-selection decision (Section 4).

After fixing the policy type (for example, resignation policy), the next thing in designing policy $\mathbf{P}$ is to stipulate the condition for $\mathbf{P}$ to become *satisfied*. This includes

1. identifying and evaluating a decision-related criterion Cr (Steps 2 and 3 in Fig. 7),
2. ordering selected peers based on Cr (Step 4 in Fig. 7), and
3. deriving and applying a decision threshold θ to determine whether $\mathbf{P}$ is satisfied or not (Steps 5 and 6 in Fig. 7).

Overall, we assume that when applied to a peer, every criterion will evaluate to a real number. Let $Cr(p^i)$ denote the value of Cr for peer p^i. Suppose that a certain performance metric q (for example, the average service request response time (SSRT)) is important in our FSP system. Then, it is a highly desirable property that the criterion Cr can be closely related to q. Specifically, if Cr increases, q will increase (or decrease) with a high probability. Intuitively, this property makes it straightforward to estimate the merit of any decision and will serve as the *guideline* for criteria identification (Step 2 in Fig. 7).

5.1 The Policy Design Process

To help understand the policy-driven decision process, the design of two resignation policies (Fig. 7) will be detailed in this section. In many situations, two important criteria are closely related to the resignation decision:

- Cr_1. The amount of computation resources (CPU cycles) actually contributed by a peer to process service requests within a unit period of time (that is, 1 second).
- Cr_2. The fraction of idle time, where idle time refers to the time period during which a peer has no service requests to process.

These two criteria are selected with the aim of reducing the average SRRT and improving the average resource utilization (RU). Roughly, if Cr_1 increases, SRRT might decrease because the peer under consideration has put more resources into processing service requests. Meanwhile, if Cr_2 decreases, RU might increase because the peer has spent more time working. Each of the two criteria is associated with a separate decision policy. Literally, we wish those peers that have contributed less computation resources and frequently remained idle to leave the service group (that is, to resign).

To actually construct the two resignation policies, we have adapted our method from a technique termed EO [9]. At the core of EO is the so-called *extremal selection mechanism* (ESM), which is used originally to study punctuated equilibrium in biology [5]. In a nutshell, ESM states that within a population of *individuals* (for example, biological creatures), the one with the lowest *fitness* should be replaced by a new individual (for example, a randomly generated individual). In many optimization problems, EO has been shown to perform well in finding desirable solutions [10]. Following the same philosophy as EO, it is convenient to view each peer p^i involved in a resignation decision as an individual. Assume that policy $\mathbf{P}_1$ uses criterion Cr_1 and policy $\mathbf{P}_2$ relies on criterion Cr_2. The fitness of peer p^i in policy $\mathbf{P}_1$ will be equal to $Cr_1(p^i)$. Its fitness in policy $\mathbf{P}_2$ will be equal to $-Cr_2(p^i)$. This definition of fitness is adopted with the expectation of removing peers that perform poorly in terms of SRRT and RU from their respective service groups. According to ESM, we have

- $\mathbf{P}_1$. A peer is allowed to resign if its Cr_1 value is the lowest among the m peers.
- $\mathbf{P}_2$. A peer is allowed to resign if its $-Cr_2$ value is the lowest among the m peers.

$\mathbf{P}_1$ and $\mathbf{P}_2$ are very rigid in their above form. It is possible that all peers in a service group will never satisfy both $\mathbf{P}_1$ and $\mathbf{P}_2$ concurrently. The reason is because Cr_1 and Cr_2 are correlated in a specific way. If a peer has devoted much of its computation resources (that is, Cr_1), it will probably have a short idle time (that is, Cr_2). However, when considering multiple peers (for example, m peers in the resignation decision), Cr_1 and Cr_2 are not consistent in meaning. A peer with a high processing capability may have a long idle time, despite of the fact that it has contributed more computation resources than other peers. On one hand, we wish peers that share more resources to stay in the group in order to maintain high performance (for example, a short SRRT). On the other hand, we also want these peers to resign if their resources are not being utilized efficiently.

In order to give every peer at least a small possibility to resign, a heuristic used in EO is further adopted. This corresponds to step 5 in Fig. 7. Use $\mathbf{P}_1$ as an example. Suppose that peer p^1 is making a resignation decision, which involves also the local information of peers $p^2, \cdots, p^m$. We can define a permutation Π as

$$Cr_1(p^{\Pi(1)}) \leq Cr_1(p^{\Pi(2)}) \leq \cdots \leq Cr_1(p^{\Pi(m)}).$$

The peer p^i with the lowest Cr_1 value is ranked 1, $\Pi(1) = i$. The peer with the highest Cr_1 value is ranked m. A probability distribution $\mathbf{Pr}$ is imposed over the ranks r as

$$\mathbf{Pr}(r) \propto r^{-\tau}, 1 \leq r \leq m$$

for a given τ. Increasing τ will result in a higher probability for lower ranks. Based on Boettcher and Percus' theoretical analysis, the value of τ can be set to $1 + [ln(N_p)]^{-1}$, where N_p is the number of peers in the FSP system [8]. At each time of applying policy $\mathbf{P}_1$, peer p^1 will select a peer p^k according to distribution $\mathbf{Pr}$. The Cr_1 value of p^k serves as a decision threshold θ that helps p^1 to judge whether its $Cr_1(p^1)$ is comparatively low. Step 6 in Fig. 7 realizes this judgment through a *stimulus-response mechanism*.

In the literature, a *stimulus-response function* (F_s) modeled after the task allocation behavior of wasps has been successfully exploited to coordinate distributed service providers [13]. In the context of policy $\mathbf{P}_1$, the stimulus is $Cr_1(p^1)$, and the response threshold is $\theta = Cr_1(p^k)$. The stimulus-response function F_s adopted in this paper is given as follows:

$$F_s\big(Cr_1(p^1)\big) = \frac{ArcTan(\delta_{P_1} \cdot (Cr_1(p^1) - \theta))}{\pi} + \frac{1}{2}. \quad (1)$$

δ_{P_1} is termed the *steepness factor* of policy $\mathbf{P}_1$. The probability that peer p^1 is allowed to resign by policy $\mathbf{P}_1$ is equal to

$1 - F_s(Cr_1(p^1))$. When δ_{P_1} is gradually increased, p^1 will become 1) more and more unlikely to resign if $Cr_1(p^1)$ is greater than θ or 2) more and more likely to resign if $Cr_1(p^1)$ is less than θ. In view of this, δ_{P_1} actually controls the influence of θ on the final decision of policy $\mathbf{P}_1$. Its impact on the system performance is further evaluated in Section 6.6.

By following the policy design methodology given in Fig. 7 and using our methods for steps 5 and 6, a policy is fully determined upon knowing 1) its decision-related criterion and 2) the total order relation defined over the criterion.

5.2 Policy Design under Multiple Performance Requirements

In real applications, it is very likely for an FSP system to offer various types of services with diverse quality-of-service (QoS) and resource requirements. QoS requirements in this paper refer mainly to the quality and nonfunctional aspect of a service. These may include response time, latency, availability, reliability, result precision, etc. As our research aims at self-organizing the group structure GS in response to the varied service demand, only *performance requirements* (for example, response time and failure rate) will be considered. In general, performance requirements are closely related to resource requirements. A peer is only capable of handling a service request provided that it has the necessary resources. In an attempt to handle an arbitrary number of performance requirements, an extension of the policy design methodology in Fig. 7 will be presented in this section.

Inspired by the system model in [19], we can easily formalize our coordination problem as follows: Each service s in an FSP system is associated with a group of performance metrics $[q_1, \ldots, q_d]$. As a basic requirement, every metric q will be evaluated to a real number. The higher the q, the better the system performs. For example, we can define q to be equal to $1/SRRT$, where $SRRT$ denotes the average SSRT. When SRRT decreases, which is commonly desirable, metric q will increase accordingly. Performance metrics are not always independent. In particular, the increase of one metric might affect another metric negatively. There must be a trade-off between different metrics. For this reason, an overall performance measure J is also defined for every service as

$$J = \sum_{i=1}^{d} \omega_i \cdot q_i, \quad \sum_{i=1}^{d} \omega_i = 1, 0 \leq \omega_i \leq 1, \tag{2}$$

where ω_i is a weight that represents the importance of different performance metrics. We can customize J by adjusting these weights. The *objective* of our coordination mechanism is to constantly improve J of every service provided by the FSP system.

Assume that there is a peer p that offers service s. The request for s comes randomly to p. In order to process these requests, peer p needs to consume its shared resources, which are characterized by a list of resource properties $[\rho_1, \ldots, \rho_r]$ (peers' local information). Each property ρ describes the shared resources from a certain perspective. For example, a property termed *processing capability* is used to measure the CPU cycles contributed by a peer (Section 4). In case that there is more than one peer hosted in the same machine, they compete with each other for CPU cycles, and a peer may temporarily become unavailable. The *availability rate* (AR) and *time-to-recover* (TTR) therefore serve as another two properties of the shared CPU cycles. TTR represents the time it takes for a peer to become available again. Other properties might be the available network bandwidth, the memory space, etc.

Notice that the performance measure of service s, J^s, is defined as the average over all the service requests processed by a group of peers that provide service s. As a result, the performance of any peer p in the group, J_p^s, can actually be considered as an estimation of J^s. Because the process to handle each request is largely determined by the resources shared by p, it is eligible to view the *expectation* of any performance metric q involved in evaluating J_p^s as a function of the property list $[\rho_1, \ldots, \rho_r]$ of p:

$$E(q) = Q([\rho_1, \ldots, \rho_r]). \tag{3}$$

Consequently, the *expectation* of J_p^s also becomes a function of $[\rho_1, \ldots, \rho_r]$ as

$$E(J_p^s) = \sum_{i=1}^{d} \omega_i \cdot Q_i([\rho_1, \ldots, \rho_r]). \tag{4}$$

For the purpose of policy design, $E(J_p^s)$ in (4) enjoys a very important property. That is, when $E(J_p^s)$ is high, J^s is also expected to be high ($E(J_p^s)$ is closely related to J^s). According to our guideline for criteria identification, it seems reasonable to treat $E(J_p^s)$ as a decision criterion for p. Policy construction becomes rather straightforward with the three rules below:

- Rule 1. Peer p is allowed to *resign* if $E(J_p^s)$ is relatively low.
- Rule 2. Peer p is allowed to be *employed* if $E(J_p^s)$ is relatively high.
- Rule 3. Peer p should *select a service group* in which it has a relatively high $E(J_p^s)$.

Based on the above rules, we can follow the steps in Section 5.1 to implement the resignation policy, employment policy, and group-selection policy. The intuition behind is to assign any peer p to the service group where it can contribute most and remove p from any service group where it will contribute less.

The result of our discussion in this section is a *general method* to deal with multiple performance requirements. As a prerequisite for its practical application, function $Q(\cdot)$, which establishes the relation between $E(q)$ and $[\rho_1, \ldots, \rho_r]$ ((3) and (4)), must be defined. In many situations, $Q(\cdot)$ may be either represented mathematically (for example, based on theoretic analysis) or obtained through low-cost learning and simulation. In our experiments to be introduced in Section 6.6, $Q(\cdot)$ is evaluated via simulations.

6 Experiments

Experiments have been conducted in an attempt to evaluate the effectiveness of the recruiting protocol and policy-based decision heuristics (that is, our coordination mechanism) described in Sections 4 and 5. These experiments were

performed in a P2P simulation system called *PeerSim*. The system configuration varies in different experiments in order to test the robustness of our coordination mechanism with respect to environment changes. In total, experiments in five different application scenarios will be presented in this section. In the first four scenarios, SRRT is considered as the main performance metric to be improved. In order to further examine the method proposed in Section 5.2, in the last application scenario, the performance measure J is redefined as a weighted sum of two performance metrics, SRRT and the failure rate. The definitions of these performance metrics, as well as other system objectives involved in our experiments, are described below:

- **SRRT.** This is obtained by averaging the response time of all service requests submitted to the FSP system within a period of time. The response time is defined as the duration from the time when the FSP system receives a service request to the time when a peer has finished processing this request.
- **Failure rate.** This is the fraction of service requests in which the processing has failed. Request handling can fail if a peer that is processing a request becomes unavailable.
- **Resource efficiency.** This is the fraction of computation resources actually being utilized to process service requests within each service group.
- **Load distribution.** This is the distribution of the service processing load among all peers in the service groups. We expect that the processing load can be evenly distributed in the FSP system, subject to the local processing capability of each peer. We use the standard deviation over the peers' idle time divided by the peers' average idle time as a measure for the distribution of the load. The measure is given in the percentage format.

In the sequel, simulation and common experimental settings will be introduced first. Experiments in the five application scenarios will be described subsequently.

6.1 The Simulation System

The peers in the simulation system are divided into *fast*, *medium*, and *slow* categories based on their *processing capability*. The processing capability of each peer follows a normal distribution. For the fast category, the mean of the normal distribution is set to 1×10^5 computations (that is, CPU cycles) per second, and the deviation is 1×10^4. For the medium category, the mean is set to 5×10^4, and the deviation is 8×10^3. For the low category, the mean is 2×10^4, and the deviation is 3×10^3. In general, there can be five times in difference between two peers. A peer is randomly assumed for one of the three categories. The fast category comprises roughly 30 percent of the system population. The medium category covers 40 percent, and the low category takes the remaining 30 percent.

The number of requests received by the FSP system for any service s^i is assumed to follow the Poisson distribution with mean μ_{s^i}. By adjusting the value of μ_{s^i}, we are able to control the demand for various services. All peers can receive service requests from users. If necessary, these requests will be redirected to peers that offer the requested service. In our simulation system, every peer has a *request queue* to hold incoming service requests. When the request queue of a peer p becomes empty, p will ask other peers in the same service group for unprocessed service requests.

TABLE 1
The Four Decision Policies Used in the Experiments

Policy name	Decision Criterion *Cr*	Order Relation ($\preceq$)	Decision Threshold and Steepness Factor
$\mathbf{P}_1$	Computation resources contributed by a peer to satisfy service requests.	$p^i \preceq p^j$ if $Cr_1(p^i) \leq Cr_1(p^j)$	θ_{p1} (determined after the order relation $\preceq$) and δ_{p1}
$\mathbf{P}_2$	The fraction of idle time.	$p^i \preceq p^j$ if $Cr_2(p^i) \geq Cr_2(p^j)$	θ_{p2} (determined after the order relation $\preceq$) and δ_{p2}
$\mathbf{P}_3$	The processing capability of a peer.	$p^i \preceq p^j$ if $Cr_3(p^i) \leq Cr_3(p^j)$	θ_{p3} (Determined after the order relation $\preceq$) and δ_{p3}
$\mathbf{P}_4$	The fraction of idle time.	$p^i \preceq p^j$ if $Cr_4(p^i) \leq Cr_4(p^j)$	θ_{p4} (Determined after the order relation $\preceq$) and δ_{p4}

The recruiting protocol is implemented to run after every 10 seconds. A peer will not have much intention to leave a service group before it has served in this group for 50 seconds (to maintain the stability of the group structure). We assume that peers have fast connections to the Internet and the time used to transfer a message between any two peers is negligible as compared to the time used to process service requests. Peers are initially assigned to provide a randomly selected service. All services are chosen with equal probability.

In many experiments, the entire simulation runs for up to 100,000 simulated seconds. Because performance metrics such as SRRT converge long before the simulation ends, we have decided to present the simulation results before 20,000 simulated seconds. This may help to show the transient process toward convergence more clearly. The simulation advances in time steps of 10 seconds. The responses for service requests received from 50 seconds onward are collected for evaluating performance metrics.

Based on our previous discussions, it is convenient to let the performance measure J be equal to $1/SRRT$ in the first four application scenarios. In Section 5.1, two policies have already been designed to govern the resignation decision. In this section, a summary of all four policies created by us to reduce SRRT is further presented in Table 1. Each criterion used in these policies is assumed to be part of a peer's local information, which we believe is fairly common in a distributed environment. Our intention is to use very simple criteria that are closely related to J to guide the peers' local decision process.

Besides resignation policies $\mathbf{P}_1$ and $\mathbf{P}_2$, policies $\mathbf{P}_3$ and $\mathbf{P}_4$ in Table 1 will handle, respectively, the employment and the group-selection decisions. According to $\mathbf{P}_3$, a peer p will have more chance to be employed if its processing capability is relatively high. $\mathbf{P}_3$ is designed to reduce SRRT, especially when fast peers are available in the FSP system. Additionally, based on policy $\mathbf{P}_4$, p will prefer to join a service group where it will have less idle time (in other words, a group where it can contribute more). In all application scenarios, a simple hand-tuning technique [13] is used to determine the value of every steepness factor δ_P. According to our experience, the performance of the simulation system does not vary considerably for different δ_P. An experiment to demonstrate this can be found in Section 6.6. With respect to each steepness factor in Table 1,

TABLE 2
Steepness Factor Values Used in the Experiments

δ_{P_1}	δ_{P_2}	δ_{P_3}	δ_{P_4}
0.8	0.4	3.6	1.8

Table 2 further lists the corresponding value used in our experiments.

6.2 Scenario 1

We first consider a relatively simple application scenario. The FSP system contains 30 peers and provides a single service s^1. μ_{s^1} is adjusted such that approximately two service requests will be received every 10 sec. The number of computations required for processing each request follows a normal distribution with mean 1×10^6 and deviation 5×10^4. Obviously, the number of peers involved in the simulation outweighs the demand for s^1. In order to improve the resource efficiency, the service group $G^1 =< \{p^i\}, s^1 >$ needs to maintain only enough peers for request processing. Meanwhile, since there are no competing service groups, peers with higher processing capabilities should be more likely to be employed by G^1 so as to reduce SRRT. Finally, it is also desirable that the processing load can be evenly distributed over the peers in group G^1.

Simulations have been performed to verify all the above expectations. τ is set to 1.3 according to the heuristic presented in Section 5.1. m and h in the recruiting protocol (Figs. 4, 5, and 6) are set to 10 and 1, respectively. Fig. 8 illustrates the change in group G^1's size during simulation. As evidenced in Fig. 8, the group size gradually decreases until it reaches 8. Fig. 9 further shows the average processing capability of the peers in group G^1 during simulation. There is a drop in the average processing capability between 10,000 and 15,000 seconds in Fig. 9. One possible explanation of this drop is that it is due to the sudden increase of service demand, which has resulted in a long request queue. In response to this demand fluctuation, group G^1 employs more peers with a relatively low processing capability. However, even at the bottom of the drop, the average processing capability is still much higher than that of medium-category peers. Under a long simulation with 100,000 simulated seconds, it is witnessed that a drop with such a time span seldom happens.

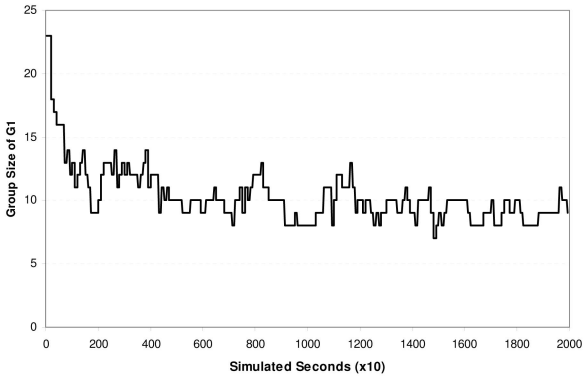

Fig. 8. The change in group G^1's size during simulation.

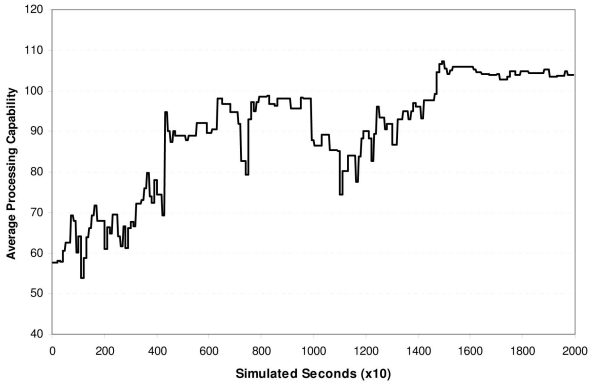

Fig. 9. Average processing capability of group G^1 during simulation.

As the average processing capability of group G^1 increases (Fig. 9), a short SRRT of 13.8 seconds is achieved. Notice that if we perform simulation without using the decision policies in Table 1, SRRT would be around 28 seconds. Moreover, in the case that group G^1 contains only fast-category peers, the best SRRT would be slightly higher than 10 seconds. During a majority time period (from 2,000 simulated seconds onward) of the simulation, the resource efficiency is maintained at a level of 0.8, and the load distribution is around 25 percent. We feel this is quite acceptable after considering the randomness involved in generating and processing service requests.

6.2.1 Comparison Experiment

Application scenario 1 introduced in this section has posed a similar problem as addressed by the resource management framework proposed by Cuenca-Acuna and Nguyen [15]. As a comparison study, we have implemented their framework within our simulation system. The essential part of the framework is to periodically use GA[1] to identify those peers that together form service group G^1. The selection of a group member in GA is guided by a fitness function, which is able to estimate the expected performance of any group configurations according to the current service demand. Conceptually, it is eligible to define the fitness function as

$$\lambda\big(\{p^i\}, \alpha \cdot \mu \cdot \nu\big), \tag{5}$$

where μ refers to the average number of service requests received per second, ν represents the average number of computations consumed by each service request. α is a positive factor such that when the total processing capability of peers in $\{p^i\}$ exceeds $\alpha \cdot \mu \cdot \nu$, group G^1 is said to outweigh the service demand. Our implementation of function λ in (5) follows exactly the fitness model in [15]. Simulation results with varied α's are summarized in Table 3.

Table 3 indicates that with different αs, GA will produce different configurations for service group G^1. As the group size increases, SRRT will decrease accordingly at the expense of reduced resource efficiency. In general, there seems to be a trade-off between SRRT and resource efficiency in the FSP system. Comparing with the results in Table 3, it is further evidenced that our coordination mechanism is able to balance the trade-off desirably due to

1. In the experiment, GA is used after every 100 seconds.

TABLE 3
Comparison Experiment Results Using the Resource Management Framework in [15]

α	SRRT (s)	Group Size	Resource Efficiency (%)
1.0	24.3	5.3	83.2%
1.5	13.5	8.9	62.5%
2.0	12.0	9.8	47.0%
2.5	11.8	10.2	38.9%

TABLE 4
Specific Settings for the Five Services Adopted in the Simulation

Service	Number of Request (50-20000)	Times of Computation (Mean)	Times of Computation (Deviation)	Group
s^1	4000	$1x10^6$	$5x10^4$	G^1
s^2	4000	$2x10^6$	$1x10^5$	G^2
s^3	4000	$1.5x10^6$	$5x10^4$	G^3
s^4	4000	$1x10^6$	$5x10^4$	G^4
s^5	4000	$5x10^5$	$5x10^4$	G^5

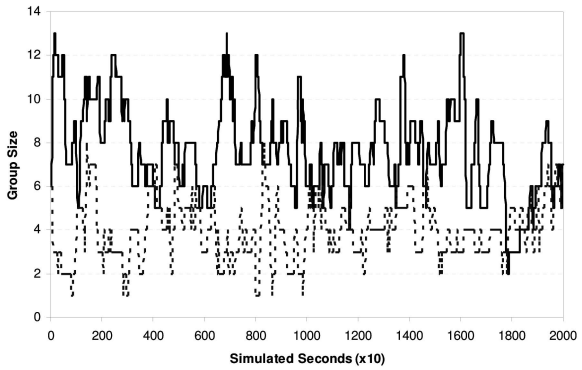

Fig. 10. Group size change for G^2 (real line) and G^5 (dashed line) during simulation.

the relatively short SRRT and high resource efficiency. Moreover, our results are achieved through the peers' local interaction without relying on any global information required by GA.

6.3 Scenario 2

In this section, we will consider an application scenario where peers are not seemingly abundant as in Section 6.2. Intuitively, groups with a high processing demand should be able to attract more peers, whereas other groups' interests may be compromised. The FSP system contains 30 peers as in Section 6.2 and offers five services. Table 4 lists the specific settings of each service.

Since there are five service groups in the simulation, m and h in the recruiting protocol now are equal to 6 and 5, respectively. Other settings remain the same as described in Section 6.2. Fig. 10 shows the group size change for G^2 and G^5 from 0 to 20,000 simulated seconds. Fig. 11 depicts the change of the total processing capability of the two service groups.

According to Figs. 10 and 11, group G^2 has a larger group size and more computational power than group G^5 during the simulation. This is due to the fact that processing a request for service s^2 will consume more computation resources than processing a request for s^5 (Table 4). Table 5 summarizes the average group size, the total processing

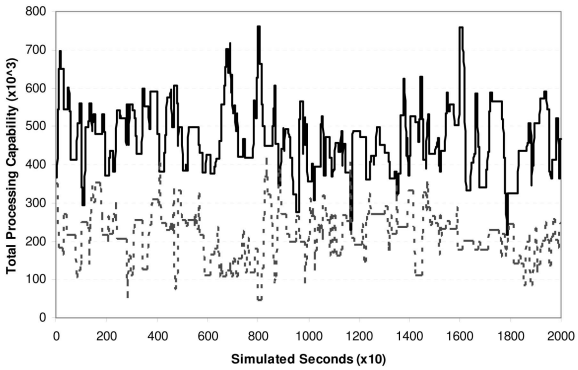

Fig. 11. Total processing capability of G^2 (real line) and G^5 (dashed line) during simulation.

TABLE 5
Average Performance of Each Service Group during Simulation

	G^1	G^2	G^3	G^4	G^5
Average Group Size	5.8	7.6	6.9	5.5	4.4
Total processing Cap. ($\times 10^3$)	297	469	400	292	215
SRRT (s)	42.1	46.6	45.6	43.4	27.0

capability, and the average response time for the five groups.

As evidenced in Table 5, the group structure is able to properly reflect the varied service processing demand via self-organization. As a result, the SRRTs of the five service groups are very close to each other. The standard deviation of SRRT is 5.6. Without using the decision policies in Section 5, this deviation may reach to 93.6.

Similar to our observations in Section 6.2, for a majority period of time during simulation, the resource efficiency of each service group stays around 0.8. Specifically, as shown in Fig. 12, there is only a small difference between the resource efficiency of separate service groups. This fact in turn implies that our coordination mechanism has achieved a fairly even distribution of load.

6.4 Scenario 3

In this section, we further test the effectiveness of our coordination mechanism when an FSP system contains more peers and offers more services. In one experiment, 200 peers are involved in providing 10 different services. The settings of each peer follow the same rule as introduced in Section 6.1. The first five services provided by the system (that is, s^1, s^2, s^3, s^4, and s^5) use the same settings as in Table 4, whereas the last five services (that is, s^6, s^7, s^8, s^9, and s^{10}) simply copy the respective settings of the first five services. Table 6 lists SRRT and average group size of the 10 service groups observed during simulation.

As shown in Table 6, despite of the fact that 200 peers were involved in the simulation, the group size for each service is only enlarged slightly as compared with the experiment results in Table 5. This is because the demand for each service is kept at the same level as the experiment in Section 6.3. Nevertheless, SRRT is considerably reduced since the competition between service groups is not as strong as in Section 6.3. More opportunities are available for each service group to employ fast-category peers.

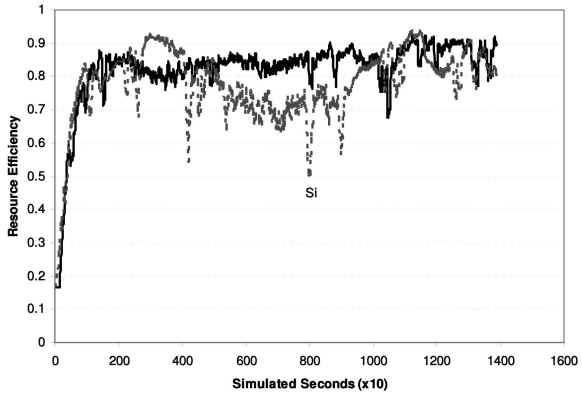

Fig. 12. Resource efficiency of G^2 (real line) and G^5 (dashed line) during simulation.

The experiment results in Table 6 suggest that our coordination mechanism *scales well* with an increasing number of services and peers. The improved opportunity of employing fast peers is exploited to reduce SRRT, whereas the group size remains compatible with the service demand. In fact, more extreme experiments are performed. In one experiment where 500 peers and 40 services are involved in the simulation, the SRRTs of those 10 services listed in Table 6 are still maintained at the same level (Table 7).

6.5 Scenario 4

In order to test the robustness of our coordination mechanism in the face of joining and leaving peers, two experiments are conducted in this section. In both of the experiments, the FSP system provides five services, as defined in Table 4. In the first experiment, there are initially 30 peers in the system. As the simulation continues, new peers will constantly join the system at a speed of one peer per 20 sec until there are 200 peers. Fig. 13 illustrates the change of SRRT with respect to service s^2 during simulation.

As evidenced in Fig. 13, newly joined peers have been effectively exploited to gradually reduce SRRT. After around 4,500 simulated seconds, SRRT cannot be further improved because all the 200 peers have joined the FSP system by that time. The final response time observed at 20,000 simulated seconds is 25.0 seconds, which is very close to the results given in Table 6.

In the second experiment in this section, there is a probability of 0.02 for peers that just resigned from their service groups to leave the system. There are 200 peers that are active before the simulation starts. Fig. 14 depicts the number of peers in the FSP system, as well as the SRRT of service s^2, during simulation.

TABLE 6
Average Performance of Each Group during Simulation

	s^1	s^2	s^3	s^4	s^5
Average Group Size	7.0	8.1	7.1	7.3	6.9
SRRT (s)	17.7	24.5	22.5	18.0	14.1
	s^6	s^7	s^8	s^9	s^{10}
Average Group Size	7.2	8.2	7.5	7.3	6.8
SRRT (s)	17.7	24.4	22.7	18.0	14.0

TABLE 7
Average Performance of Each Service Group during a Simulation with 500 Peers and 40 Services

	s^1	s^2	s^3	s^4	s^5
Average Group Size	6.1	8.4	7.5	6.2	5.2
SRRT (s)	18.1	30.7	26.5	18.2	13.9
	s^6	s^7	s^8	s^9	s^{10}
Average Group Size	6.0	8.4	7.6	5.9	5.2
SRRT (s)	18.3	30.6	27.1	18.3	13.7

Since peers may permanently leave the FSP system, the number of peers during simulation is continuously decreasing. By the time the simulation is terminated, there are only 41 peers left in the system. As the system shrinks its size, the SRRT of service s^2 is gradually increasing but at a much slower pace. In fact, SRRT has only been increased by about 8 sec as fast-category peers are more reluctant to resign and consequently leave the system according to the policies in Table 1. Based on the results in Figs. 13 and 14, we believe that our coordination mechanism is good at handling joining/leaving peers.

6.6 Scenario 5

In this section, a new performance metric, namely, the failure rate, will be considered together with SRRT. Different from the four application scenarios discussed previously, the resources shared by any peer in the FSP system are characterized by three properties, which are the

1. processing capability ρ_1,
2. AR ρ_2, and
3. TTR ρ_3.

As usual, ρ_1 refers to the number of computations allowed per second. ρ_2 measures the probability for a peer p to become unavailable temporarily. As ρ_2 approaches 1, peer p will have more chances to remain available. Even if peer p is unavailable, it can still stay in a service group. However, no service requests will be accepted and processed by peer p anymore. The processing of any service request will fail at the time when peer p becomes unavailable. As time passes, peer p will become available again. The average time for peer p to recover is indicated by property ρ_3. Notice that it is possible to use other properties

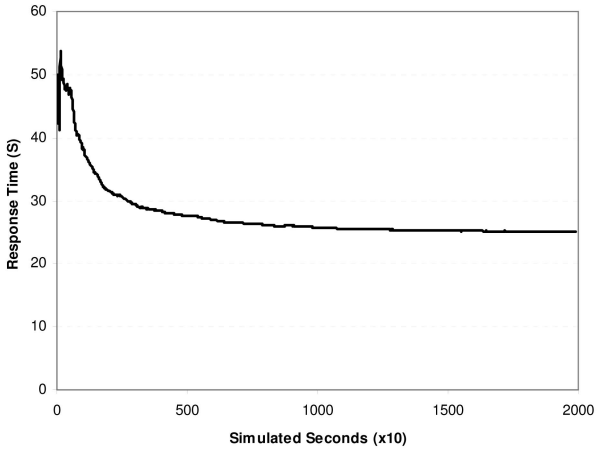

Fig. 13. SRRT of service s^2 during simulation.

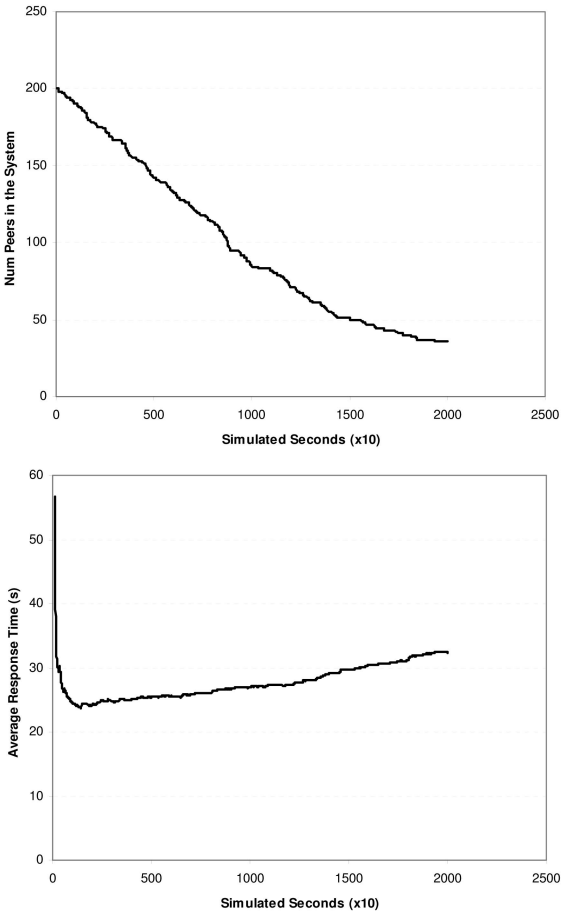

Fig. 14. The number of peers in the system and the response time for service s^2 during simulation.

and models to describe shared resources, depending on specific application requirements. In this section, only properties ρ_1, ρ_2, and ρ_3 will be utilized for the purpose of demonstrating the policy design method in Section 5.2.

Similar with the preceding experiments, the FSP system contains three categories of peers. Properties ρ_2 and ρ_3 of each peer are assumed to follow normal distributions. In order to highlight the interaction between SRRT and the failure rate, the AR (ρ_2) of fast-category peers will be considerably lower than that of medium-category peers. The specific settings for each category of peers can be found in Table 8.

Following the policy design method in Section 5.2, exactly one resignation policy, one employment policy, and one group-selection policy are created. We use δ_{res}, δ_{emp}, and δ_{grp} to denote, respectively, the steepness factors of the three policies. Function $Q(\cdot)$ (3) is evaluated through low-cost simulations. For each peer p, the simulation starts

TABLE 8
Resource Property Settings for Each Category of Peers

Peer category	Mean ρ_2	Deviation ρ_2	Mean ρ_3 (s)	Deviation ρ_3
Fast	0.80	0.03	30	5
Medium	0.99	0.01	30	5
Slow	0.95	0.03	30	5

TABLE 9
SRRT and Failure Rate of Service s^2

Experiment	SRRT (s)	Failure Rate (%)
Experiment A	49.2	12.9%
Experiment B	45.6	19.6%

with the assumption that a fraction of the service requests have been assigned to p. The processing of every service request is estimated according to the computation speed ρ_1 of p. Request processing might fail with a probability that is equal to $1 - \rho_2$. Finally, the predicted value of SRRT or the failure rate (that is, $Q(\cdot)$) is obtained by averaging all handled service requests during simulation.

Experiments have been performed with an FSP system that contains 50 peers and offers five services, as defined in Table 4. Our experiments show that by adjusting the weight ω_1 associated with $1/SRRT$ (2), we can have different performance results. Table 9 specifically compares the failure rate and SRRT of service s^2 in two experiments, A and B. The $SRRT$ in experiment A $(\omega_1 = 0.8)$ is less important than that in experiment B $(\omega_1 = 0.95)$.

Due to the low AR (ρ_2) of fast-category peers, in order to reduce the failure rate, service groups need to employ more medium-category peers. Therefore, the SRRT in experiment A is longer than the SRRT in experiment B (Table 9). Fig. 15 helps to demonstrate the effectiveness of our coordination mechanism by showing the performance measure J associated with s^2 in experiment B. Experiments with varied system configurations are conducted. Overall, we find that policies designed via the method in Section 5.2 are as effective as those policies applied in the preceding four sections.

Experiments are also carried out in an attempt to understand how sensitive the results are to chosen steepness factors δ_p. Using policies designed in this section, one particular experiment is conducted with the FSP system introduced in Section 6.2, which offers a single service s^1. Throughout the experiment, δ_{res} and δ_{grp} are fixed at 0.8 and 3.5, respectively. Fig. 16 shows the group size and SRRT of service s^1 when δ_{emp} has different values. As indicated in Fig. 16, after increasing δ_{emp} from 0.5 to 10, the group size

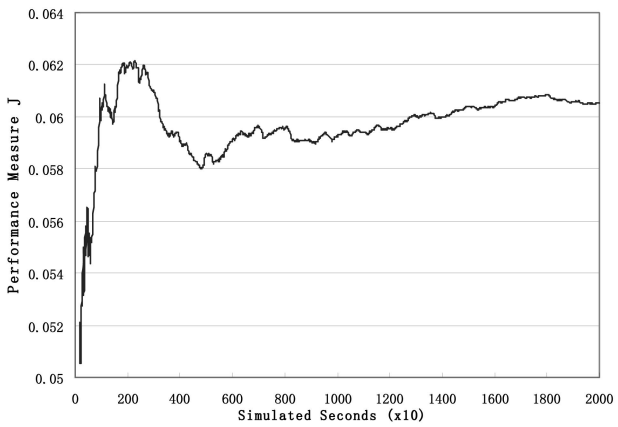

Fig. 15. The change in the performance measure J of service s^2 during simulation.

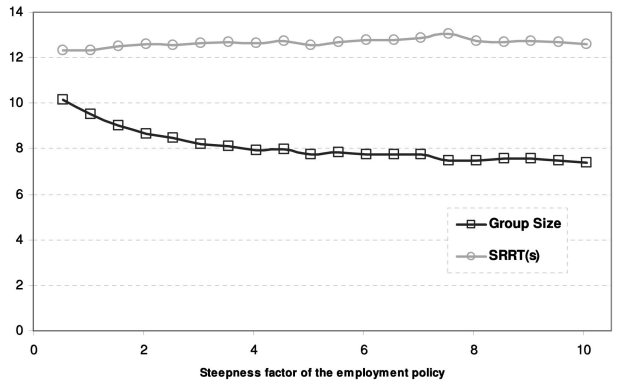

Fig. 16. The group size and SRRT of service s^1 for different values of δ_{emp}.

will shrink to about 2.3, whereas the performance metric SRRT does not vary significantly. Based on this and other experiments we performed, we believe that the FSP system is not very sensitive to steepness factors.

7 Conclusion

In this paper, we targeted on the coordination problem that arises in the FSP systems where distributed peers contribute their computation resources to offer domain-specific services. The research focuses on the question of how to adjust the service group structures with respect to varying service demands through self-organization. We have presented our solution under the scope of a coordination mechanism.

A labor-market model was proposed in order to establish an analogy between the FSP system and a simplified social recruiting structure. A recruiting protocol was further built on top of the model to regulate the peers' interaction through a recruiting process. The protocol followed a policy-driven decision architecture. A general methodology was introduced in this paper for policy design. Each policy is mapped to a decision-related criterion, which, in turn, establishes a total order relation over the peers involved in the decision-making process. We have identified the problem that policies may contradict with each other to a certain extent. A general heuristic inspired by EO was adopted to solve this problem. A stimulus-response mechanism was also utilized to make the decision process more adjustable. In real applications with diverse performance and resource requirements, an extra method for policy design was further introduced based on the estimated performance measure.

Experiments in a simulated P2P environment were conducted under five application scenarios. The system configuration involved in the simulation varied in different experiments in order to test the robustness of our coordination mechanism. The experiment results showed that our mechanism was effective in terms of improving resource efficiency, reducing SRRT and the failure rate, and balancing the load distribution. Based on the experiment results, we believe that our coordination mechanism might become a practical solution in real-life FSP applications.

In this paper, it was assumed that peers are cooperative in nature. They are willing to contribute all their shared resources to the processing of incoming service requests. However, the problem of cooperation and incentives in P2P systems is a major concern in the literature. In order to encourage cooperation, it is necessary to extend our coordination mechanism with effective incentive mechanisms. For example, an incentive mechanism that gives any peer a higher priority of accessing services offered by other peers when it has processed many service requests might encourage peers to contribute [14].

It is to be noticed that the purpose of our work is the initial exploration of coordination problems in FSP environments. We do not address issues such as the degree of fault tolerance, cooperation, security, etc. These and other problems will be left here as future work.

References

[1] Kazaa Media Desktop, http://www.kazaa.com/, 2001.
[2] BitTorrent, http://bitconjurer.org/, 2003.
[3] M. Adler, R. Kumar, K. Ross, D. Rubenstein, D. Turner, and D.D. Yao, "Optimal Peer Selection in a Free-Market Peer-Resource Economy," *Proc. Second Workshop Economics of Peer-to-Peer Systems,* 2004.
[4] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys,* vol. 36, no. 4, pp. 335-371, 2004.
[5] P. Bak and K. Sneppen, "Punctuated Equilibrium and Criticality in a Simple Model of Evolution," *Physics Rev. Letters,* vol. 71, pp. 4083-4086, 1993.
[6] F. Banaei-Kashani, C.C. Chen, and C. Shahabi, "WSPDS: Web Services Peer-to-Peer Discovery Service," *Proc. Int'l Symp. Web Services and Applications,* 2004.
[7] *Grid Computing: Making the Global Infrastructure a Reality,* F. Berman, G. Fox, and A.J.G. Hey, eds. John Wiley & Sons, 2003.
[8] S. Boettcher and M. Grigni, "Jamming Model for the Extremal Optimization Heuristics," *J. Physics A: Math. and General,* pp. 1109-1123, 2002.
[9] S. Boettcher and A.G. Percus, "Optimization with Extremal Dynamics," *Physics Rev. Letters,* vol. 23, no. 4, pp. 5211-5214, 2001.
[10] S. Boettcher and A.G. Percus, "Extremal Optimization: An Evolutionary Local-Search Algorithm," *Computational Modeling and Problem Solving in the Networked World: Interfaces in Computer Science and Operations Research,* H.K. Bhargava and N. Ye, eds., pp. 61-77, Kluwer Academic Publishers, 2003.
[11] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Computational Power Grids," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '00),* 2000.
[12] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 35, no. 3, pp. 373-384, 2005.
[13] V.A. Cicirello and S.F. Smith, "Wasp-Like Agents for Distributed Factory Coordination," *Autonomous Agents and Multi-Agent Systems,* vol. 8, pp. 237-266, 2004.
[14] B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. First Workshop Economics of Peer-to-Peer Systems,* 2003.
[15] F.M. Cuenca-Acuna and T.D. Nguyen, "Self-Managing Federated Services," *Proc. 23rd IEEE Symp. Reliable Distributed Systems,* 2004.
[16] Y. Fu, Z. Dong, and X. He, "An Approach to Web Services Oriented Modeling and Validation," *Proc. Int'l Workshop Service-Oriented Software Eng. (IW-SOSE '06),* pp. 81-87, 2006.
[17] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," *Proc. IEEE INFOCOM,* 2004.
[18] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1989.
[19] X. Gu and K. Nahrstedt, "Distributed Multimedia Service Composition with Statistical QoS Assurances," *IEEE Trans. Multimedia,* vol. 8, no. 1, pp. 141-151, 2006.
[20] X. Gu and K. Nahrstedt, "On Composing Stream Applications in Peer-to-Peer Environments," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 8, pp. 824-837, Aug. 2006.
[21] X. Gu, K. Nahrstedt, and B. Yu, "SpiderNet: An Integrated Peer-to-Peer Service Composition Framework," *Proc. 13th IEEE Int'l Symp. High-Performance Distributed Computing (HPDC '04),* pp. 110-119, 2004.

[22] D. Hausheer and B. Stiller, "PeerMart: The Technology for a Distributed Auction-Based Market for Peer-to-Peer Services," *Proc. 40th Int'l Conf. Comm. (ICC '05),* 2005.
[23] D. Hausheer and B. Stiller, "PeerMint: Decentralized and Secure Accounting for Peer-to-Peer Applications," *Proc. Fourth Int'l IFIP-TC6 Networking Conf. (Networking '05),* pp. 40-52, 2005.
[24] C.H. Hsu, T.L. Chen, and G.H. Lin, "Grid Enabled Master Slave Task Scheduling for Heterogeneous Processor Paradigm," *Proc. Fourth Int'l Conf. Grid and Cooperative Computing (GCC '05),* pp. 449-454, 2005.
[25] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," *Proc. Fifth Int'l Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS '06),* pp. 915-922, 2006.
[26] D. Kondo, M. Taufer, C.L. Brooks, H. Casanova, and A. Chien, "Characterizing and Evaluating Desktop Grids: An Empirical Study," *Proc. 18th IEEE/ACM Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2004.
[27] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Comm. Surveys and Tutorials,* vol. 7, no. 2, pp. 72-93, 2005.
[28] O. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Service Discovery in Agent-Based Pervasive Computing Environments," *Mobile Networks and Applications,* vol. 9, pp. 679-692, 2004.
[29] D.M. Reeves, M.P. Wellman, J.K. Mackie-Mason, and A. Osepayshvili, "Exploring Bidding Strategies for Market-Based Scheduling," *Decision Support Systems,* vol. 39, no. 1, pp. 67-85, 2005.
[30] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms,* 2001.
[31] G. Shao, F. Berman, and R. Wolski, "Master/Slave Computing on the Grid," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 3-16, 2000.
[32] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking,* vol. 11, no. 1, pp. 17-32, 2003.
[33] K. Sycara, M. Paolucci, A. Anolekar, and N. Srinivasan, "Automated Discovery, Interaction and Composition of Semantic Web Services," *Web Semantics,* vol. 1, no. 1, 2003.
[34] R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 34, no. 6, pp. 2426-2438, 2004.
[35] B. Yang and H. Garcia-Monlina, "Efficient Search in Peer-to-Peer Networks," *Proc. 22nd IEEE Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[36] B. Yu and M.P. Singh, "A Social Mechanism of Reputation Management in Electronic Communities," *Proc. Fourth Int'l Workshop Cooperative Information Agents (CIA '00),* pp. 154-165, 2000.
[37] J. Yu, S. Venugopal, and R. Buyya, "A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and Their Services," *J. Supercomputing,* vol. 36, no. 1, pp. 17-31, 2006.

Gang Chen received the PhD degree in computer science from Nanyang Technological University, Singapore, in 2006. He is currently a teaching fellow with the School of Electrical and Electronics Engineering, Nanyang Technological University. His main research interests are multiagent systems, peer-to-peer networks, machine learning, and evolutionary algorithms.

Chor Ping Low received the PhD degree in computer science from the National University of Singapore in 1994. He is currently an associate professor with the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore. His current research interests include network performance modeling and analysis, combinatorial optimization, and engineering informatics. He is a member of the IEEE Computer Society and the ACM.

Zhonghua Yang received the PhD degree in computing and information technology from Griffith University, Brisbane, Australia. He is currently an associate professor with the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore. His earlier career included working for the Singapore Institute of Manufacturing Technology; Griffith University; University of Queensland, St. Lucia, Australia; University of Alberta, Canada; and Imperial College, London. He spent a significant part of his career with the Chinese aerospace industry. His research interests include grid/distributed computing, semantic Web services, and multiagent systems. He is a senior member of the IEEE.