Agent-Based Cloud Computing

Kwang Mong Sim, *Senior Member*, *IEEE*

Abstract—Agent-based cloud computing is concerned with the design and development of software agents for bolstering cloud *service discovery, service negotiation*, and *service composition*. The significance of this work is introducing an agent-based paradigm for constructing software tools and testbeds for cloud resource management. The novel contributions of this work include: 1) developing *Cloudle*: an agent-based search engine for cloud service discovery, 2) showing that agent-based negotiation mechanisms can be effectively adopted for bolstering cloud service negotiation and *cloud commerce*, and 3) showing that agent-based cooperative problem-solving techniques can be effectively adopted for automating cloud service composition. *Cloudle* consists of 1) a service discovery agent that consults a *cloud ontology* for determining the similarities between providers' service specifications and consumers' service requirements, and 2) multiple *cloud crawlers* for building its database of services. *Cloudle* supports three types of reasoning: *similarity reasoning, compatibility reasoning*, and *numerical reasoning*. To support *cloud commerce*, this work devised a complex cloud negotiation mechanism that supports parallel negotiation activities in interrelated markets: a cloud service market between consumer agents and broker agents, and multiple cloud resource markets between broker agents and provider agents. Empirical results show that using the complex cloud negotiation mechanism, agents achieved high utilities and high success rates in negotiating for cloud resources. To automate cloud service composition, agents in this work adopt a *focused selection contract net protocol* (*FSCNP*) for dynamically selecting cloud services and use service *capability tables* (*SCTs*) to record the list of cloud agents and their services. Empirical results show that using *FSCNP* and *SCTs*, agents can successfully compose cloud services by autonomously selecting services.

Index Terms—Cloud computing, multiagent systems, software agent, service discovery, service composition, negotiation, resource management

1 Introduction

Whereas many existing works in cloud computing focus on the development of infrastructures and tools for pooling together computational resources, this work complements and supplements existing works in cloud computing by introducing "agent-based cloud computing"—applying agent-based approaches to managing cloud computing infrastructures.

1.1 Cloud Computing

A cloud is a large group of interconnected computers that extends beyond a single company or enterprise [1], [2]. The applications and data served by the cloud are accessed via the Internet by a broad group of users across multiple enterprises and platforms. A cloud computing system consists of a collection of interconnected and virtualized computers dynamically provisioned as one or more unified computing resource(s) through negotiation of service-level agreements (*SLAs*) between providers and consumers [3]. In cloud computing platforms, resources need to be dynamically (re)configured and aggregated via virtualization [3] and consumers' requirements can potentially vary over time and amendments need to be accommodated.

1.2 Agent-Based Computing

An agent is a computer system that is capable of autonomous (independent) actions, that is, deciding for itself and figuring out what needs to be done to satisfy its design objectives [5]. A multiagent system consists of a number of agents, which interact with one another [5]. To successfully interact, agents require the ability to cooperate, coordinate, and negotiate with each other. Cooperation is the process when several agents work together and draw on the broad collection of their knowledge and capabilities to achieve a common goal. Coordination is the process of achieving the state in which actions of agents fit in well with each other. Negotiation is a process by which a group of agents communicate with one another to try to come to a mutually acceptable agreement on some matter.

1.3 Agent-Based Cloud Computing

Some of the essential characteristics of cloud computing include resource pooling and resource sharing. In clouds, computing resources are pooled to serve multiple consumers, and applications and data are available to and shared by a broad group of cross-enterprise and cross-platform users. Resource pooling and sharing involve 1) combining resources through cooperation among cloud providers, 2) mapping, scheduling, and coordination of shared resources, and 3) establishment of contracts between providers and consumers. In agent-based cloud computing, cooperation, negotiation, and coordination protocols of agents are adopted to automate the activities of resource pooling and sharing in clouds.

Supporting autonomous resource mapping and dealing with changing requests accentuate the need for cloud resource management systems that are capable of continuously managing the resource reservation process by monitoring current service requests, amending future service requests, and autonomously adjusting schedules and prices to accommodate dynamically changing resource demands [3]. Sim [4] proposed that software agents are

• *The author is with the School of Computing, The University of Kent, Chatham Maritime, Kent, ME4 4AG, United Kingdom. E-mail: prof_sim_2002@yahoo.com.*

Manuscript received 1 Dec. 2010; revised 4 July 2011; accepted 25 Aug. 2011; published online 10 Oct. 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org and reference IEEECS Log Number TSCSI-2010-12-0154. Digital Object Identifier no. 10.1109/TSC.2011.52.

appropriate tools for autonomously managing cloud resources. Whereas consumers need to make decisions to select suitable providers and negotiate with providers to achieve "ideal" service contracts, providers need to make decisions for selecting appropriate requests to accept and execute depending on the availability of resources, both current and future demands for services, and existing service obligations. Since agents are capable of making decisions when carrying out tasks on behalf of their users, and interacting with other agents through negotiation, cooperation, and coordination, all the above-mentioned challenges provide the motivations for adopting autonomous agents to allocate resources amid dynamically changing resource demands.

Agent-based cloud computing is concerned with the design and development of software agents for bolstering cloud service discovery, service negotiation, and service composition [4], [6], [7], [8]. Appendix 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2011.52, shows an overview of applying agent-based approaches to building software tools and testbeds for some of the phases of a cloud service life cycle. A cloud service life cycle [9] consists of: service requirements, service discovery, service negotiation, service composition, and service consumption. In the service requirements phase, consumers detail the functional requirements (functions and tasks that a service should provide), technical requirements (e.g., hardware and operating systems), and budgetary requirements (acceptable service cost). The service discovery phase consists of searching for cloud services that match consumers' functional, technical, and budgetary requirements. The service negotiation phase consists of message exchanges between consumers and brokers, and between brokers and providers for establishments of service-level agreements. In the service composition phase, a broker combines a set of services from multiple providers, and delivers the combined service as a single virtualized service to a consumer. In the service consumption phase, the service is delivered to the consumer.

The contributions of this work are:

1. introducing an agent-based paradigm for designing software tools and testbeds for managing cloud resources,
2. designing and developing Cloudle—an agent-based search engine for supporting cloud service discovery,
3. showing that negotiation protocols in multiagent systems can be effectively adopted for bolstering cloud service negotiation and cloud commerce, and
4. showing that cooperative problem-solving paradigms in multiagent systems can be effectively adopted for automating cloud service composition.

1.4 Agent-Based Cloud Search Engine

The challenge in the service discovery phase is to run a query against the cloud services registered in the search engine's database by matching consumers' functional, technical, and budgetary requirements. When building a general search engine (e.g., Google), one only needs to consider the issue of searching for webpages that contain concepts in a user's query. The problem of building a search engine for cloud services is *more complex* because one needs to search for services that satisfy three types of requirements. The search engine in this work employs a service discovery agent (*SDA*) that consults a cloud ontology for determining the similarities between providers' functional and technical specifications of services and consumers' functional and technical requirements (Section 2).

1.5 Negotiation Agents and Agent-Based Cloud Commerce

The challenge in cloud service negotiation is to establish *SLAs* between consumers and brokers, and between brokers and service providers. Whereas e-commerce negotiation mechanisms involve two types of participants (buyers and sellers) in *only one* market and participants are not allowed to breach contracts, the problem of devising a complex negotiation mechanism for cloud commerce is much more complex because a complex cloud negotiation mechanism specifies parallel negotiation activities among *three* types of participants (consumers, brokers, and providers) in *multiple interrelated markets* and participants are *allowed to breach contracts* by paying penalty fees. This work devises a complex cloud negotiation mechanism by using negotiation strategies and protocols in multiagent systems [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20] as the basic building blocks (Section 3).

1.6 Agent-Based Cloud Service Composition

The challenge in cloud service composition is to dynamically put together a set of services provided by multiple service providers to form a single unified service to be delivered to a consumer. Various providers need to work together and draw upon each other's service capabilities. To automate cloud service composition, this work adopts 1) a *focused selection contract net protocol* (*FSCNP*) for dynamically selecting cloud services and 2) *service capability tables* (*SCT*s) to record the list of cloud agents and their services.

2 Cloud Search Engine and Cloud Crawlers

Whereas preliminary ideas of *Cloudle* were reported in [23], [24], [25], this work presents a *new* architecture of *Cloudle* (Fig. 1) consisting of a *service discovery agent*, a *cloud ontology*, a *database of cloud services*, multiple *cloud crawlers* (Section 2.3), and a *web interface*.

In the previous design [23], [24], [25], *Cloudle* relied solely on cloud service providers to register their services in the database of cloud services. In the new *Cloudle* architecture, in addition to allowing service providers to register their services in Cloudle's database, cloud crawlers are also used to build and maintain Cloudle's database (Section 2.3)—this is a novel feature that enhances the previous architecture of Cloudle reported in [23], [24], [25].

Consumers enter their queries for cloud services using Cloudle's web interface (Appendices 2 and 3, available in the online supplemental material). The inputs to Cloudle consist of a consumer's functional, technical, and budgetary requirements for a cloud service. The *SDA* has four submodules:

1. a query processor,
2. a service reasoning module,
3. a price and time slot matching module, and
4. a service rating module.

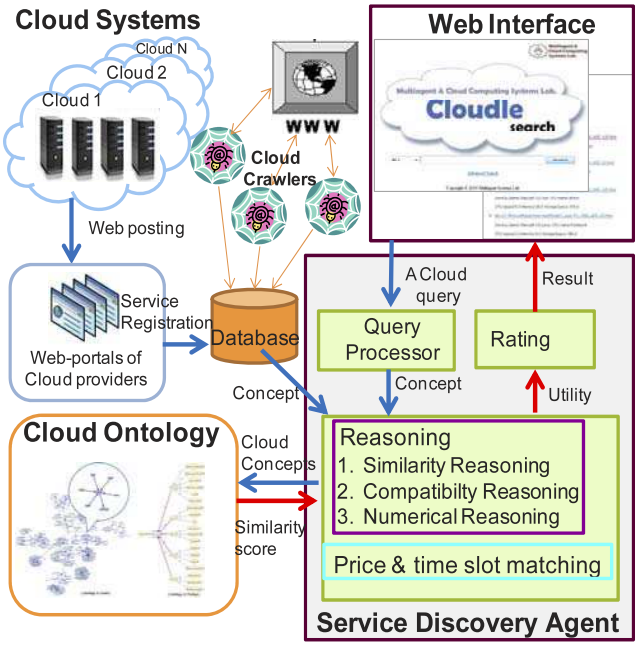


Fig. 1. The Cloudle architecture.

Using the query processor, the *SDA* extracts essential keywords such as cloud concepts and the price and schedule specifications from consumers' queries.

The *SDA* consults a cloud ontology (Appendix 4, available in the online supplemental material) to reason about the similarity between a consumer's functional and technical requirements and providers' functional and technical specifications of services. A cloud ontology consists of a set of cloud computing concepts and the interrelationships among cloud concepts to facilitate the reasoning about cloud services. In this work, the cloud ontology consists of over 400 concepts. At the top level of the cloud ontology [26], cloud services are generally classified into

1. Infrastructure as a service (*IaaS*) which provides computational resources (e.g., Amazon's EC2 provides VMs),
2. Data as a service (*DaaS*) which allows users to store data at remote disks (e.g., Amazon's S3),
3. Software as a service (*SaaS*) which delivers special-purpose software, remotely accessed via the Internet (e.g., Salesforce's *CRM*),
4. Platform as a service (*PaaS*) that supplies cloud developers with programming-level environments (e.g., Google App Engine provides a python runtime environment and API for applications to interact with Google's cloud runtime environment), and
5. Communication as a service (*CaaS*) that provides dedicated bandwidth, network security, encryption, and network monitoring.

Using the cloud ontology, the *SDA* carries out: 1) similarity reasoning, 2) compatibility reasoning, and 3) numerical reasoning (Section 2.1). Additionally, using the price and time slot module, the *SDA* determines the level of matching between the price and schedule constraints of both consumers and providers (Section 2.2). Using the service rating module, the service provided by each provider is rated according to the similarity between the functional and technical specifications of both the consumer and the provider. The output consists of a list of cloud services ordered in terms of the matching similarities between the functional, technical, and budgetary constraints specified by the consumer and service providers (Appendix 5, available in the online supplemental material).

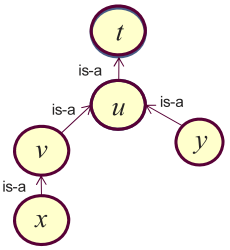


Fig. 2. A simple general ontology.

2.1 Service Reasoning

Given that Cloudle needs to satisfy three types of requirements: 1) functional, 2) technical, and 3) budgetary, sometimes, it may be difficult to find services that will exactly match these three types of requirements.

Similarity reasoning is designed to increase the chance of finding relevant alternatives of a service. For instance, if an exact matching service is beyond a consumer's price range, other similar services within the price range may be suggested. An intuitive way to determine the degree of similarity between two concepts x and y is to determine how much x and y share in common. In similarity reasoning, the *SDA* determines the similarity between x and y by counting their common reachable nodes. Let $\alpha(x)$ (respectively, $\alpha(y)$) be the set of nodes upwards reachable from x (respectively, y) including x (respectively, y) itself. For example, in the graph of general ontology (Fig. 2), $\alpha(x) = 4$ and $\alpha(y) = 3$. Let $\alpha(x) \cap \alpha(y)$ be the number of reachable nodes shared by x and y. $\alpha(x) \cap \alpha(y)$ is a measure of the common features between x and y. In Fig. 2, $\alpha(x) \cap \alpha(y) = 2$. There are several ways to define a function for determining the degree of similarity between x and y [27]. One way to measure the degree of similarity between x and y is to measure the number of common features between x and y from the perspective of x as follows:

$$sim(x, y) = \left| \frac{\alpha(x) \cap \alpha(y)}{\alpha(x)} \right|.$$

Another way is to measure the number of common features between x and y from the perspective of y as follows:

$$sim(y, x) = \left| \frac{\alpha(x) \cap \alpha(y)}{\alpha(y)} \right|.$$

In Fig. 2, $\mathrm{sim}(x, y) = 2/4$ and $\mathrm{sim}(y, x) = 2/3$, which are different because $\mathrm{sim}(y, x) > sim(x, y)$. To take both measure methods into consideration, a generalized similarity function for x and y can be defined by taking the weighted average of both measure methods as follows:

$$sim(x, y) = \rho \left| \frac{\alpha(x) \cap \alpha(y)}{\alpha(x)} \right| + (1 - \rho) \left| \frac{\alpha(x) \cap \alpha(y)}{\alpha(y)} \right|, \quad (1)$$

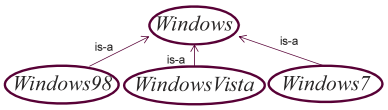


Fig. 3. A partial Windows ontology.

where $\rho \in [0, 1]$. An example is given in Appendix 6, available in the online supplemental material. In Appendix 6, ρ is set to 0.5 so that both measure methods are given equal consideration. $sim(x, y) = 0$ means that x is totally not similar to y, and $sim(x, y) = 1$ means that x is fully similar to y.

Compatibility reasoning is designed to compare two sibling nodes in a cloud ontology, e.g., determining the compatibility between two different versions of a software. The rationale for devising compatibility reasoning is because using similarity reasoning, the *SDA* cannot distinguish between two different versions of a software concept. For example, in the partial *Windows* ontology in Fig. 3,

$$\begin{aligned}\alpha(Windows98) = \alpha(WindowsVista) = \alpha(Windows7) = 2,\\ \alpha(Windows98) \cap \alpha(WindowsVista)\\ = \alpha(Windows98) \cap \alpha(Windows7)\\ = \alpha(WindowsVista) \cap \alpha(Windows7) = 1,\\ \text{and } sim(Windows98, WindowsVista)\\ = sim(Windows98, Windows7)\\ = sim(WindowsVista, Windows7) = 1/2.\end{aligned}$$

Hence, using similarity reasoning, the *SDA* cannot reason about the differences among sibling concepts.

Since two sibling nodes representing two different versions of a software will have a high degree of similarity, but differ only in terms of their chronological ordering, the function for measuring the compatibility of two concepts x and y consists of: 1) measuring the degree of similarity between x and y, and 2) differentiating between x and y in terms of their chronological ordering as follows:

$$compat(x, y) = sim(x, y) + \frac{(\mu^{|c_x - c_y|})}{\theta}, \tag{2}$$

where c_x and c_y are the label values of x and y respectively, and *sim(x,y)* is defined in (1). c_x and c_y represent the chronological orderings of different versions of a software. In (2), *sim(x,y)* is a coarse-grain measurement because x and y being different versions of a software will have a high degree of similarity and $(\mu^{|c_x - c_y|})/\theta$ is a fine-grain measurement because x and y will have a small degree of difference. It is also intuitive to think that the degree of difference will depend on the chronological ordering. For instance, compared to *WindowsVista*, *Windows95* will have less features compatible with *Windows7*. In the previous design of Cloudle [23], [24], [25], the compatibility function was defined as follows:

$$compat(x, y) = sim(x, y) + \frac{(0.8^{|c_x - c_y|})}{10}. \tag{3}$$

Equation (2) in this work generalizes (3) in [23], [24], [25]. This is because one can set different values for $0 < \mu < 1$ and $1 \leq \theta < \infty$. The most essential component in $(\mu^{|c_x - c_y|})/\theta$ is the term $|c_x - c_y|$. When $|c_x - c_y|$ is large (respectively, small), x and y are less (respectively, more) compatible. An example is given in Appendix 7, available in the online supplemental material. In Appendix 7, μ and θ are set to 0.8 and 10, respectively.

In numerical reasoning, the *SDA* reasons about the similarity between two numeric concepts (e.g., CPU speed and memory size) relative to the maximum and minimum values as follows:

$$Sim(a, b, c) = 1 - \left|\frac{a - b}{Max_c - Min_c}\right|, \tag{4}$$

where a and b are numeric values and c is a concept. An example is given in Appendix 8, available in the online supplemental material.

2.2 Price and Time Slot Matching

In matching price and time slot, the *SDA* attempts to search for providers with cloud services that have available time slots that *match the specified time slots* of consumers and with *small price difference* between the acceptable prices of the provider and consumer. A utility function $\boldsymbol{U}$ is used to evaluate the level of matching for each potential match M_k between the respective prices and schedules of a consumer and a provider. The utility of M_k is given as follows:

$$U(M_k) = w_P \cdot U^P\big(P_{\min}^{prov}\big) + w_T \cdot U^T\big(T_S^{prov}\big),$$

where w_P and w_T are a consumer's preference for more matching service prices and more matching time slots, respectively, and $w_P + w_T = 1$. $U^P(P_{\min}^{prov})$ is a consumer's price utility defined as follows:

$$U^P\big(P_{\min}^{prov}\big) = \big(P_{\max}^{cons} - P_{\min}^{prov}\big)/P_{\max}^{cons},$$

where $P_{\max}^{cons}$ is the maximum acceptable price for a consumer, and $P_{\min}^{prov}$ is the minimum acceptable price for a provider. $U^P(P_{\min}^{prov}) \to 0$ if $P_{\min}^{prov} \to P_{\max}^{cons}$ since there will be little or no room for negotiation of price. $U^P(P_{\min}^{prov}) \to 1$ if $P_{\min}^{prov} \to 0$ since there will be plenty of room for negotiation. $U^T(T_S^{prov})$ is a consumer's time slot utility defined as follows:

$$U^T\big(T_S^{prov}\big) = \big(T_S^{cons} \cap T_S^{prov}\big)/T_S^{cons},$$

where T_S^{cons} is the range of possible time slots that the consumer expects to reserve a time slot to utilize a service, and T_S^{prov} is the range of time slots that a service is available. It is assumed that T_S^{cons} is greater than the actual time slot (end time − start time) that a consumer will utilize a service. $U^T(T_S^{prov}) \to 1$ (respectively, $U^T(T_S^{prov}) \to 0$) if there are more (respectively, less) overlap between T_S^{cons} and T_S^{prov}.

2.3 Cloud Crawlers

Many parallel threads of the cloud crawler are deployed to gather information about cloud service providers. The architecture of a cloud crawler is shown in Fig. 4.

It consists of a crawling agent, a *URL* filter softbot (a software agent), and a database agent. The crawling agent traverses the *WWW* to extract webpages that are relevant to cloud computing services. Starting from a root *URL* that is predefined by a user, the crawler agent traverses websites by following hyperlinks. As the crawling agent visits a

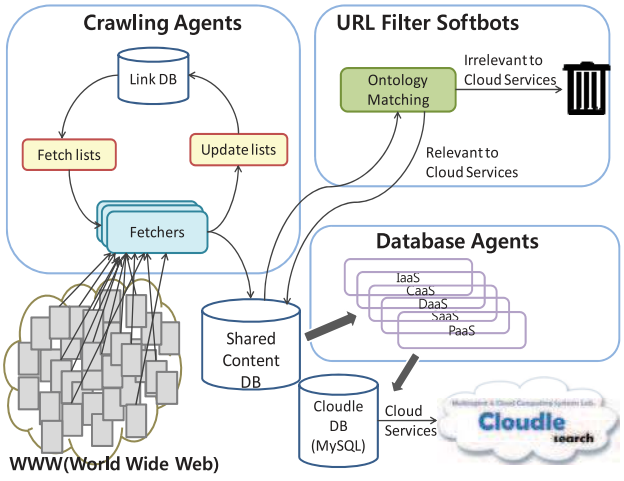


Fig. 4. An architecture of a cloud crawler.

website, its fetcher module downloads a copy of the webpage, then examines the contents and extracts the link data (consisting of *URLs* and hyperlinks) and the contents of the webpage by parsing the downloaded document (stripping away the html tags). The fetcher module stores the extracted contents in a shared content memory and updates the link data in the crawling agent's link database. Using an ontology of cloud concepts that is built using protégé, the *URL* filter softbot examines the contents of webpages stored in the shared content memory and determines if the contents are relevant to cloud services. The *URL* filter softbot scans the texts in each document that are stored in the shared content database to search for cloud concepts in the ontology. A document that contains more cloud concepts receives a higher score. The database agent examines relevant documents in the shared content memory by extracting the name of the service provider, service type, price, and technical specifications such as *CPU* speed and *RAM* capacity. All such information together with the *URL* of the webpage will be stored in *Cloudle*'s database.

3 Cloud Commerce and Negotiation Agents

In a cloud business model, consumers pay service providers for consumption of computing capabilities. It was noted in [3] that a *market-oriented approach* for managing cloud resources is necessary for regulating the supply and demand through flexible and dynamic pricing.

Cloud market. A market model for trading cloud resources described in [3] consists of resource/service providers, consumers, and brokers. Brokers purchase resource capacities from providers, compose multiple resources from different providers into bundled services, then sublease the unified services to consumers. Each broker accepts requests from multiple consumers and each consumer can also submit its service requests to many brokers. Consumers, brokers, and providers are bound to service contracts through *SLAs* that specify the details of the service to be provided agreed upon by all cloud participants, and the penalties for violating the expectations. Such a market provides an infrastructure for 1) connecting disparate clouds from different providers, 2) allowing consumers to select appropriate services that can satisfy their resource requirements, and 3) allowing service providers to effectively set the prices of their computing resources based on market conditions, consumer demand, and their current levels of resource utilization. To date, even though service providers have inflexible pricing for cloud resources, it is envisioned that a market infrastructure described in [3] will enable variable cloud service pricing based on market conditions. Sim [6], [8] suggests that automated negotiation is an appropriate economic model to facilitate flexible pricing of cloud services based on varying demand from consumers and varying supply from providers.

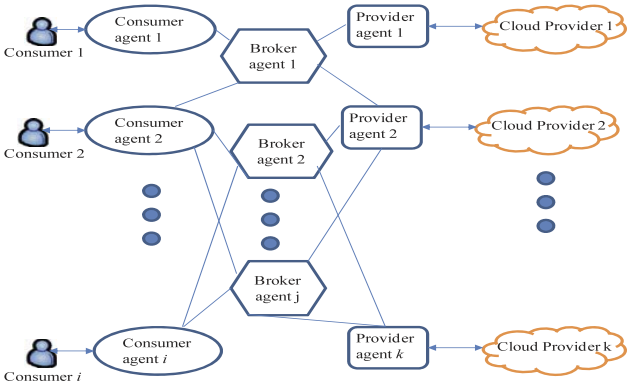


Fig. 5. An agent-based testbed for simulating cloud commerce.

Agent-based testbed. Fig. 5 shows the design of an agent-based testbed for simulating cloud commerce [6], [8]. It consists of provider agents and consumer agents acting on behalf of resource providers and consumers, respectively, and a set of broker agents. Broker agents accept service requests from consumer agents, purchase resources from provider agents, dynamically compose a collection of resources to satisfy consumer agents' requirements, then sublease the service to consumer agents. In doing so, broker agents need to carry out negotiation activities in two types of markets. In a *cloud service market*, broker agents negotiate with consumer agents for mutually acceptable terms to establish *SLAs* for satisfying service requirements from consumers. In *cloud resource markets*, broker agents negotiate with resource providers for reserving resources.

Cloud negotiation model. Fig. 6 shows a cloud negotiation mechanism for facilitating the negotiation activities 1) between consumer agents and broker agents and 2) between broker agents and provider agents. Since each broker agent can accept requests from many consumer agents and each consumer agent can also submit its requirements and requests to many broker agents, it is envisioned that a many-to-many negotiation model be adopted for negotiation between consumer agents and broker agents. Since a cloud service may be dynamically composed using multiple types of cloud resources, each broker agent can potentially negotiate in multiple types of cloud resource markets with multiple groups of cloud providers that provide different types of cloud resources. Hence, a concurrent one-to-many negotiation mechanism is adopted to facilitate concurrent negotiation activities between broker agents and different groups of provider agents.

3.1 Multilateral Service Negotiation

For the many-to-many negotiation between consumer agents and broker agents, a *market-driven negotiation strategy* [16], [17], [18], [19], [20] and a *bargaining-position-estimation*

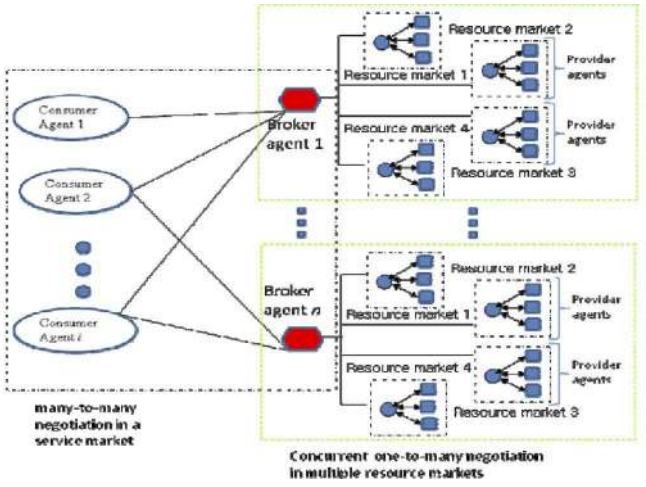


Fig. 6. A cloud negotiation model.

strategy [13] are used to determine the amounts of concessions.

Negotiation protocol. The following negotiation protocol [12], [14] is used for specifying the multilateral negotiation activities between consumer and broker agents:

- Negotiation proceeds in a series of rounds.
- Adopting Rubinstein's alternating offers protocol [29], a pair of consumer and broker agents negotiates by making proposals in alternate rounds.
- Multiple consumer-broker agent pairs can negotiate deals simultaneously.
- When an agent makes a proposal, it proposes a deal from their space of possible deals (e.g., consisting of the most desirable price, the least desirable (reserve) price, and those prices in between). Typically, an agent proposes its most preferred deal initially.
- If no agreement is reached, negotiation proceeds to the next round. At every round, an agent determines its amount of concession using the strategy described below.
- Negotiation between two agents terminates 1) when an agreement is reached, or 2) with a conflict when one of the bargaining agents' deadline is reached.
- An agreement is reached if a consumer agent CA_1 and a broker agent BA_1 propose deals P^1_{CA} and P^1_{BA}, respectively, such that either 1) $U(P^1_{CA}) \geq U(P^1_{BA})$ or 2) $U(P^1_{BA}) \geq U(P^1_{CA})$, where P^1_{CA} and P^1_{BA} represent the buying and selling prices of a cloud service, respectively, and U is a utility function.

Negotiation strategy. In a cloud service market where consumers compete for computing services and brokers compete to provide services, a market-oriented approach for regulating the supply and demand of cloud services is appropriate. To model dynamic pricing of cloud services, consumer and broker agents adopt a *market-driven approach* when making concessions similar to that in [16], [17], [18], [19], [20]. In [16], [17], [18], [19], [20], a *market-driven agent* (*MDA*) determines the appropriate amounts of concessions using a combination of three negotiation functions: time ($\boldsymbol{T}$) function, opportunity ($\boldsymbol{O}$) function, and competition ($\boldsymbol{C}$) function.

Time function. Since consumers are generally sensitive to deadlines in acquiring computing services, and deadlines may also affect brokers' scheduling and composition of services, it is intuitive to consider time when formulating the negotiation decision functions. A consumer agent's time-dependent concession-making strategies can be classified into: 1) *conservative* (maintaining the initial price until an agent's deadline is almost reached), 2) *conciliatory* (conceding rapidly to the reserve price), and 3) *linear* (conceding linearly) [16], [17], [18], [19], [20]. Let IP_{CA} and RP_{CA} be the initial and reserve prices of a consumer agent, respectively. Based on its time-dependent concession making function, the consumer agent's price proposal at negotiation round t is given as follows:

$$\mathrm{P}_{CA}(t) = IP_{CA} + \left(\frac{t}{\tau_{CA}}\right)^{\lambda_{CA}} (RP_{CA} - IP_{CA}),$$

where τ_{CA} is the consumer agent's deadline for acquiring a service, $0 < \lambda_{CA} < \infty$ is the concession making strategy. Three classes of strategies are specified as follows: *Conservative* ($\lambda_{CA} > 1$), *Linear* ($\lambda_{CA} = 1$), and *Conciliatory* ($0 < \lambda_{CA} < 1$). Details are given in [16], [17], [18], [19], [20].

Opportunity function. Since a consumer agent can submit service requests to multiple broker agents and a broker agent also receives requests from many consumers, both consumer and broker agents should also be programmed to consider *outside options*. The $\boldsymbol{O}$ function in [16], [17], [18], [19], [20] determines the amount of concession based on 1) trading alternatives (i.e., outside options) and 2) differences in utilities generated by the proposal of an agent and the counter-proposal(s) of its opponent(s). When determining opportunity, it was shown in [16], [19] that if there is a large number of trading alternatives, the likelihood that an agent's opponent proposes a bid/offer that is potentially close to the agent's own offer/bid may be high. For instance, if there is a large number of broker agents in the cloud service market, then there may be a higher chance that one or more broker agents propose prices that are close to the proposed price of a consumer agent. Nevertheless, it would be difficult for an agent to reach a consensus if none of the so many options are viable (i.e., there are large differences between the proposal of the agent and the counter-proposals of all its opponents). On this account, the $\boldsymbol{O}$ function in [16], [17], [18], [19], [20] determines the probability of reaching a consensus on an agent's own term by considering 1) trading alternatives (i.e., the outside options), and 2) differences between its proposal and the proposals of each of its opponent (i.e., the viability of each option). The general idea is that if the probability of reaching a consensus on its own terms is *high* (respectively, *low*), an agent should make a *smaller* (respectively, *larger*) amount of concession. Details for deriving the probability of reaching a consensus are given in [16], [19] and are omitted here due to space limitation.

Competition function and market rivalry. Since *both* consumers compete for services and brokers compete to provide services, market rivalry and competition should also be modeled. Furthermore, different degrees of competition need to be considered since both demand for and supply of services may vary. However, in [16], [19],

the C function determines the amount of competition of an *MDA* by considering the number of competitors and the number of available options in the market. In a market with m consumers and n brokers, a consumer agent C_1 has $m-1$ competitors $\{C_2,\ldots,C_m\}$ and n trading partners $\{B_1,\ldots,B_n\}$. The probability that C_1 is *not* the most preferred trading partner of *any* $B_j \in \{B_1,\ldots,B_n\}$ is $(m-1)/m$. Hence, the probability that C_1 is *not* the most preferred trading partner of *all* $B_j \in \{B_1,\ldots,B_n\}$ is $[(m-1)/m]^n$. In general, the probability that C_1 is considered the *most preferred* trading partner by at *least one* of $B_j \in \{B_1,\ldots,B_n\}$ is

$$C(m,n) = 1 - [(m-1)/m]^n,$$

where m and n are the numbers of consumer agents (including C_1) and broker agents, respectively.

In a cloud service market, whereas a consumer may have information about the number of brokers or providers providing the services it requires, it may not have knowledge of the number of consumers competing for the same type of service because consumers generally do not broadcast their requests to other consumers.

Bargaining position and incomplete information. In the absence of the knowledge of the number of consumers competing for the same type of service, one of the possible ways to model market rivalry and competition in negotiation is to consider an agent's bargaining position B_p (called the *BP*-estimation (*BPE*) strategy) [13]. In a *favorable market* (respectively, *unfavorable market*), a consumer agent is in an *advantageous* (respectively, *disadvantageous*) B_p because there are *more* (respectively, *fewer*) broker agents providing cloud services and *fewer* (respectively, *more*) consumers competing for cloud services. In a *balanced market*, a consumer is in a generally *neutral* B_p as the supply of cloud services is not significantly more than the demand for services. Since a consumer agent does not know the number of its competitors, one method for estimating B_p at each negotiation round t is to consider the concession patterns of each broker agent. If broker agents are making relatively *larger* (respectively, *smaller*) concessions, then it is likely that the consumer agent is in a relatively *favorable* (respectively, *unfavorable*) B_p. Similarly, the B_p of a broker agent can also be estimated by considering the concession patterns of each consumer agent.

A consumer agent's B_p can be determined as follows. (A broker agent's B_p can also be determined in a similar way). Let $\Delta^i_{BA}(t) = \mathrm{P}^i_{BA}(0) - \mathrm{P}^i_{BA}(t)$ be the difference between the *initial* and the *current* proposals of each broker agent BA_i, and $\delta^i_{BA}(t) = \mathrm{P}^i_{BA}(t-1) - \mathrm{P}^i_{BA}(t)$ be the difference between BA_i's proposals in the *previous* and *current* rounds. Let $B_p(t)$ be a consumer agent's B_p at round t. $B_p(t)$ is derived by averaging the ratio of 1) the amount of concession in the current round $\delta^i_{BA}(t)$ and 2) the average amount of concession in the previous t rounds $\Delta^i_{BA}(t)/t$. More formally,

$$B_P(t) = \underset{i}{avg}\left(\frac{t\cdot\delta^i_{BA}(t)}{\Delta^i_{BA}(t)}\right).$$

If $B_P(t) \ll 1$, the consumer agent is more likely to be in a disadvantageous bargaining position (e.g., the consumer is in an *unfavorable market*). If $B_P(t) \ll 1$, it is likely that many broker agents are making smaller concessions at round t, i.e., the average of $\delta^i_{BA}(t)$ is likely to be relatively smaller.

If $B_P(t) \gg 1$, then there are generally many broker agents making larger concessions, and the consumer agent is more likely to be in an advantageous bargaining position (e.g., the consumer agent is in a *favorable market*).

Empirical result. Experiments were carried out in [30] to evaluate the performance of agents adopting 1) the time-dependent strategy (considering *only* the time function), 2) the *MDA* strategy, and 3) the *BPE* strategy in consumer-favorable, balanced, and consumer-unfavorable markets. In the experiments, different consumer-to-broker ratios are used to simulate different market types. To simulate consumer-favorable (respectively, consumer-unfavorable) markets, consumer-to-broker ratios of 1:3, 1:5, and 1:10 (respectively, 3:1, 5:1, and 10:1) were used and a consumer-to-broker ratio of 1:1 is used to simulate a balanced market. A consumer agent's utility is used as a performance measure and a consumer agent's utility function is defined as follows:

$$U_{CA}(P_C) = u_{\min} + \left(\frac{RP_{CA} - P_C}{RP_{CA} - IP_{CA}}\right),$$

where IP_{CA} and RP_{CA} are the consumer agent's initial and reserve prices, P_C is the agreement price of both the consumer and broker agents, and u_{min} is the minimum utility that a consumer agent receives for reaching a deal at its reserve price. For the purpose of experimentation, the value of u_{min} is defined as 0.1. A consumer agent receives a utility of zero if it cannot reach an agreement with any broker agent before its deadline. Fig. 7 shows the utility of consumer agents adopting all three negotiation strategies for consumer-to-broker ratios of 1:10, 1:5, 1:3, 1:1, 3:1, 5:1, and 10:1. It can be seen from Fig. 7 that consumer agents adopting the *BPE* and *MDA* strategies generally achieved significantly higher utilities than consumer agents adopting the time-dependent strategy. Consumer agents adopting the *BPE* and *MDA* strategies can better respond to different market conditions because they do not make excessive (respectively, inadequate) amounts of concessions in favorable (respectively, unfavorable) markets.

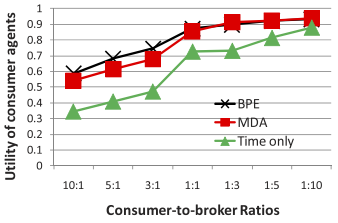


Fig. 7. Negotiation outcomes in a cloud service market.

Even though for some consumer-to-broker ratios, consumer agents adopting the *BPE* strategy did not achieve higher utilities than consumer agents adopting the *MDA* strategy (and it was proven in [16] that agents adopting *MDA* negotiate optimally), the *BPE* strategy does *not* require agents to have complete knowledge of the numbers of consumers and brokers in the cloud service market.

An analysis of MDA and BPE strategies. Whereas [30] provides empirical evidence comparing *MDA* with *BPE* under consumer-to-broker ratios of 1:10, 1:5, 1:3, 1:1, 3:1, 5:1, and 10:1, this work extends and generalizes the empirical results in [30] by analyzing the behaviors of agents adopting

the *MDA* and *BPE* strategies under extremely large and extremely small consumer-to-broker ratios as follows.

For *MDA*, to model competition from a consumer agent's perspective, let m be the number of consumers and n be the number of brokers. Since each consumer agent has $m-1$ competitors, let m' denotes $m-1$.

Hence, $C(m,n) = 1 - [(m-1)/m]^n$ can be rewritten as

$$C(m,n) = 1 - (m'/(m'+1))^n.$$

It can be shown that $C(m,n) \to 1$ as $m'/n \to 0$ as follows:

$$\lim_{\frac{m'}{n}\to 0} C(m,n) = 1 - \left(\frac{m'}{m'+1}\right)^n = 1 - \left(\frac{\frac{m'}{n}n}{m'+1}\right)^n = 1.$$

Similarly, it can also be shown that $C(m,n) \to 0$ as $m'/n \to \infty$ as follows:

$$\lim_{\frac{m'}{n}\to \infty} C(m,n) = 1 - \left(\frac{m'}{m'+1}\right)^n = 1 - \left(1 \Big/ \left(1+\frac{1}{m'}\right)^{m'}\right)^{\frac{n}{m'}}.$$

Since $\lim_{m'\to\infty}(1+\frac{1}{m'})^{m'} = e$, $(1+\frac{1}{m'})^{m'}$ is finite. As $m'/n \to \infty$ it follows that $n/m' \to 0$. Therefore,

$$\lim_{\frac{m'}{n}\to \infty} C(m,n) = 1 - \left(1 \Big/ \left(1+\frac{1}{m'}\right)^{m'}\right)^{0} = 1 - 1 = 0.$$

For *BPE*, the analyses are as follows.

As $m'/n \to 0$, there will be *fewer* consumer agents competing for services and *more* broker agents providing services, which means that consumer agents will be in an increasingly more favorable market and broker agents will be in an increasingly more unfavorable market. Hence, broker agents are increasingly more likely to make larger amounts of concessions, and $\delta^i_{BA}(t) \to \infty$. Hence, it follows that $B_p(t) \to \infty$.

As $m'/n \to \infty$, there will be *more* consumer agents and *fewer* broker agents. Broker agents will be in an increasingly more favorable market. Hence, broker agents are increasingly more likely to make smaller amounts of concessions, and $\delta^i_{BA}(t) \to 0$. Hence, it follows that $B_p(t) \to 0$.

In *MDA*, $C(m,n) \to 1$ as $m'/n \to 0$, a consumer agent faces little or no competition. In *BPE*, $B_p(t) \to \infty$ as $m'/n \to 0$, a consumer agent has the *best* B_p. Hence, as $m'/n \to 0$, a consumer agent adopting *MDA* and a consumer agent adopting *BPE* will make little or no concession. Consequently, they are both more likely to achieve high utilities. As $m'/n \to \infty$, $C(m,n) \to 0$, a consumer agent adopting *MDA* faces extremely stiff competition, and since $B_p(t) \to 0$, a consumer agent adopting *BPE* has the *worst* B_p. Hence, they will make very large amounts of concessions. In summary, as $m'/n \to 0$ (respectively, $m'/n \to \infty$), then $C(m,n) \to 1$ and $B_p(t) \to \infty$ (respectively, $C(m,n) \to 0$, and $B_p(t) \to 0$), and it follows that the utility of a consumer approaches 1 (respectively, 0).

3.2 Concurrent Cloud Resource Negotiation

For concurrent negotiation of multiple *SLAs*, a concurrent negotiation protocol adapted from [10] is adopted. Fig. 8 shows a concurrent negotiation mechanism of a broker agent for establishing multiple *SLAs* for a collection of cloud resources. It consists of a *coordinator* which coordinates the

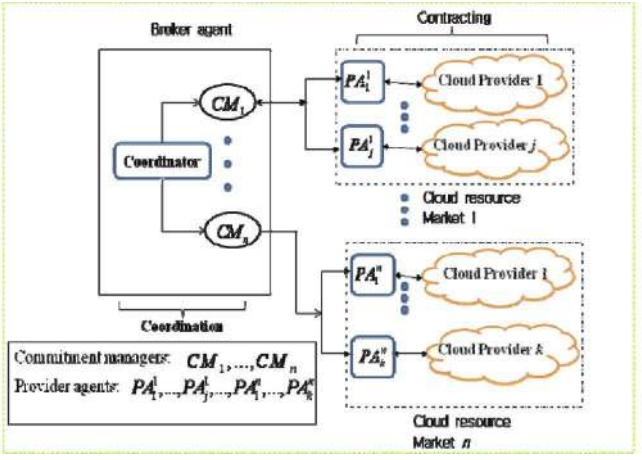


Fig. 8. Concurrent negotiation of *SLAs*.

parallel negotiation activities for acquiring n different types of cloud resources in n different cloud resource markets. In each cloud resource market, a broker agent establishes an *SLA* by negotiating simultaneously with multiple provider agents for one type of cloud resource. Furthermore, both broker and provider agents can be freed from a contract (i.e., an agent can decommit a contract) by paying penalty fees to their opponents [31], [32]. The reasons for allowing decommitments are as follows: 1) if a broker agent cannot acquire *ALL* its required resources before its deadline, it can release those resources acquired so that providers can assign them to other broker agents, and 2) it allows a broker agent that has already reached an intermediate contract for a resource to continue to search for better deals before the entire concurrent negotiation terminates. In negotiating for one type of cloud resource in a resource market, there is a commitment manager that manages both commitments and decommitments of (intermediate) contracts. In summary, two algorithms are needed for the concurrent negotiation mechanism: 1) an algorithm for establishing *SLAs* and managing commitments and decommitments of contracts (see Algorithm 1) and 2) an algorithm for coordinating the parallel negotiation activities (see Algorithm 2).

Contracting and SLAs. At each negotiation round, the commitment manager determines whether to accept the proposed offers from the provider agents or whether to renege on an intermediate contract (to break an existing contract and take up a new and more favorable contract by paying a penalty fee). However, it is inefficient for a broker agent to simply accept all acceptable proposals from provider agents and select the best proposal from them because it may be forced to pay a large amount of penalty fees for reneging on many deals. Since a cloud resource can be requested by multiple broker agents simultaneously, a provider agent can renege on an intermediate contract established with a broker agent. A broker agent determines if each provider agent's proposal $\mathrm{P}^i_j(t)$ is acceptable by first estimating the chance that a provider agent PA^i_j will renege on a contract, and compute the expected payoff of $\mathrm{P}^i_j(t)$ taking into account the chance that PA^i_j may break a contract. The general idea in estimating the reneging probability of PA^i_j is that if a broker agent reaches a tentative agreement with PA^i_j at a price that is much *lower* (respectively, *higher*) than the average price of all provider

agents, then there is a *higher* (respectively, *lower*) chance that PA_j^i will renege on the contract. Computing the expected utility of $\mathrm{P}_j^i(t)$ considers *both* the outcomes that 1) PA_j^i will renege on a contract and 2) PA_j^i will abide by the contract. $\mathrm{P}_j^i(t)$ is acceptable to a broker agent if it generates an expected utility that is higher than the utility of the broker agent's own price proposal for the resource. Algorithm 1 [10] specifies the steps for commitment management.

Algorithm 1. Commitment management

At each negotiation round t, do the following:

1. Estimate the reneging probability of each provider agent PA_j^i
2. Compute the expected utility of each provider agent's proposal $\mathrm{P}_j^i(t)$
3. Determine if each $\mathrm{P}_j^i(t)$ is acceptable
4. If there are proposals that are acceptable then
   - i. the broker agent sends a request for contracts to all corresponding provider agents
   - ii. waits for the confirmation of contract from each PA_j^i
5. If the broker agent receives one or more confirmation of contracts then
   - the broker agent accepts the contract that generates the highest expected utility

else

- the broker agent revises its proposal by making concession.

Coordination. In designing an algorithm for coordinating concurrent negotiations for establishing multiple *SLAs*, three factors are essential for a broker agent: 1) successfully establishing all *SLAs* for the required set of cloud resources needed for composing a service requested by a consumer agent, 2) obtaining the cheapest possible price for the collection of resources, and 3) establishing all *SLAs* rapidly. Since the dynamic configuration of a personalized collection of resources requires the establishment of *SLAs* between a broker agent and multiple cloud resource provider agents, the failure of any one-to-many negotiation for establishing one *SLA* will result in the failure of the entire concurrent negotiation. Hence, ensuring a high negotiation success rate is most pertinent. This paper adopts a *utility-oriented coordination (UOC) strategy* [10] for coordinating concurrent multiple one-to-many negotiations. In the *UOC* strategy, an agent always prefers higher utility when it can guarantee a high success rate. Coordination of the concurrent negotiation activities generally consists of 1) predicting the change in expected payoff in each one-to-many negotiation, and 2) deciding whether the consumer agent should proceed with or terminate the entire concurrent negotiation. Algorithm 2 [10] illustrates the steps for coordination in *UOC*.

Algorithm 2. Coordination

At each negotiation round t, do the following:

For each resource R_i,

1. The commitment manager CM_i determines if the proposal $\mathrm{P}_j^i(t)$ of each provider agent PA_j^i for resource R_i is acceptable for the broker agent (i.e., $\mathrm{P}_j^i(t)$ falls into the *agreement zone* $[IP_{BA}^i, RP_{BA}^i)$ of the broker agent)
2. If $\mathrm{P}_j^i(t)$ is acceptable for the broker agent, it will be placed into an acceptable list for R_i
3. If any acceptable list is empty then
   - the coordinator cannot complete the concurrent negotiation

else

- i. **predict the change in utility** in each one-to-many negotiation in the next round $t+1$ using the information on negotiation status supplied by each CM_i
- ii. **decide whether to terminate or proceed** with the concurrent negotiation based on the prediction of the change in utility in $t+1$ for each one-to-many negotiation.

Empirical results. Experiments were carried out in [10] to compare the *UOC* strategy with the *patient coordination strategy* (*PCS*) in [33] for coordinating n concurrent one-to-many negotiations in n cloud resource markets. In *PCS*, a broker agent terminates all concurrent one-to-many negotiations when it acquires all required resources without considering time constraint. Since there may be different supply-and-demand patterns for different types of cloud resources, for a broker agent acquiring n types of cloud resources, the market types for each cloud resource market can be *broker-favorable* (fewer brokers acquiring cloud resources and more providers supplying cloud resources), *balanced* (equal number of brokers and providers), or *broker-unfavorable* (more brokers and fewer providers). To compare *UOC* with *PCS*, six types of n cloud resource markets are simulated:

1. *all* n cloud resource markets are *favorable*,
2. *most* of the n cloud resource markets are *favorable*,
3. *all* n cloud resource markets are *balanced*,
4. *most* of the n cloud resource markets are *balanced*,
5. *all* n cloud resource markets are *unfavorable*, and
6. *most* of the n cloud resource markets are *unfavorable*.

The following three performance measures were used.

1. *Utility*. The utility of a broker agent (U_{BA}) is determined as follows:

$$U_{BA} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(U_{BA}^i(P^i) - \Gamma^i), & \\ & \text{if all Cloud resources are obtained,} \\ 0, & \text{otherwise,} \end{cases}$$

   where U_{BA}^i is the utility of a broker agent obtained by establishing a contract for cloud resource R_i at the price of P^i, and Γ^i is the total amount of penalty fee that the broker agent should pay for R_i.
2. *Success rate*. A concurrent negotiation is considered successful if a broker agent can successfully negotiate for *all* of n types of cloud resources. Otherwise, the concurrent negotiation is considered unsuccessful.
3. *Negotiation speed*. The negotiation speed of a broker agent is determined as follows:

$$S_{BA} = t_{BA}/\tau_{BA},$$

   where t_{BA} is the total number of rounds taken to complete negotiation and τ_{BA} is the broker agent's deadline. $0 \le S_{BA} \le 1$, such that $S_{BA} \to 1$ (respectively, $S_{BA} \to 0$) implies that a broker agent achieves

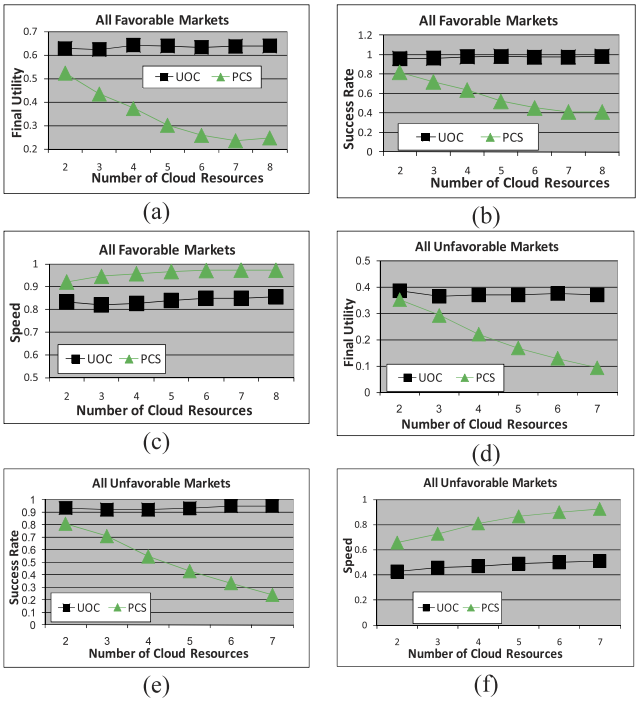


Fig. 9. Empirical results of concurrent negotiation in multiple markets.

a slower (respectively, faster) negotiation speed because it completes the entire concurrent negotiation near (respectively, long before) its deadline.

Space limitation precludes all results from being presented here and Figs. 9a, 9b, 9c, 9d, 9e, and 9f only show the graphs for the empirical results in n cloud resource markets that are *all favorable* and *all unfavorable*. These represent the extreme cases when *all* the required cloud resources are comparatively more plentiful and *all* the required cloud resources are comparatively scarcer, respectively.

From Figs. 9a, 9b, 9c, 9d, 9e, and 9f, it can be seen that broker agents adopting *UOC* achieved significantly higher utilities, higher success rates, and faster speed than broker agents adopting *PCS*. Broker agents adopting *UOC* achieved a higher success rate because it is relatively *more* difficult for a broker agent to meet the terminating condition in *PCS*. It is more difficult to have *all* CM_i in a situation where each CM_i has a provider j offering a proposal $\mathrm{P}_j^i(t)$ that generates an expected utility that is higher than the utility of the broker agent's own price proposal for the cloud resource. Broker agents adopting *UOC* can achieve faster negotiation speed than agents adopting *PCS* because broker agents adopting *UOC* can potentially terminate the entire concurrent negotiation before all CM_i's reach agreements (in the sense of having a provider's proposal that is better than or equal to its own proposal). Furthermore, since *UOC* takes into consideration the utility changes in future negotiation rounds, broker agents adopting *UOC* is more likely to achieve higher utilities than agents adopting *PCS*.

4 Agent-Based Cloud Service Composition

Service composition in cloud computing generally requires

1. coordination and interaction among cloud participants (consumers, brokers, and providers),
2. automation of service selection,
3. dynamic (re)configuration of distributed and parallel services, and
4. dealing with incomplete information about the existence and location of cloud providers and their services.

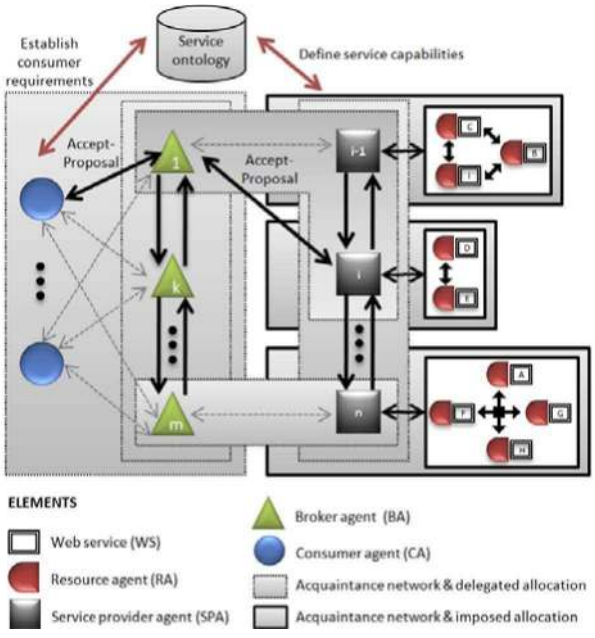


Fig. 10. An agent-based testbed for simulating cloud service composition.

The novelty of this work is proposing the ideas of

1. using service capability tables (*SCTs*) to enable agents to record the cloud services provided by other agents in the cloud systems,
2. devising a focused selection contract net protocol (*FSCNP*) for specifying the interactions of cloud agents,
3. evaluating the performance of cloud agents equipped with *SCTs* that contain different degrees of knowledge of the service capabilities of other agents, and
4. analyzing the number of messages exchanged among agents in *FSCNP* in the worst case and showing that *FSCNP* enhances the efficiency of classical *contract net protocol* (*CNP*) [21].

In *FSCNP*, agents focus on selecting relevant cloud services by consulting *SCTs*, thereby reducing the number of messages exchanged among cloud agents.

4.1 Agent-Based Testbed

An agent-based testbed for bolstering cloud service composition was implemented using Java and JADE framework. The testbed (Fig. 10) consists of web services (*WSs*), resource agents (*RAs*), service provider agents (*SPAs*), broker agents (*BAs*) and consumer agents (*CAs*).

Web services. A *WS* is an interface to a software application or a resource (e.g., database, storage space) that can be remotely accessed.

Resource agent. An *RA* manages and controls access to a web service, i.e., an *RA* is a service wrapper. An *RA* accepts requests from an *SPA* or other *RAs* to fulfill a service

requirement and transmits its service output to the requesting agent.

Service provider agent. An *SPA* manages a service provider's resources by controlling and organizing *RAs*. An *SPA* delegates some of the service requests it receives to *RAs*. See Appendix 9, available in the online supplemental material. *RAs* managed by the same *SPA* share a common set of objectives and they can delegate tasks among each other. For example, an *SPA* can even delegate some *RAs* to carry out decomposition of large tasks and allow these *RAs* to delegate subtasks to other *RAs* for processing. An *SPA* may also interact with other *SPAs*. To fulfill a service requirement, an *RA* may need the services or resources of other *RAs* that are not administered by its *SPA*. Hence, an *RA* makes a service requests through its *SPA*, which in turn contacts other *SPAs* for acquiring the services or resources. For instance, a computing service provider may need additional storage space for an unusually large amount of results derived from processing some tasks. In this case, an *RA* requiring additional storage space makes the requests through its *SPA*, which in turn searches for and collaborates with a storage *SPA* to acquire more storage space.

Broker agent. A *BA* is a mediator between *CAs* and *SPAs*. A *BA* composes a set of resources from multiple *SPAs*, and provides a single virtualized service to *CAs*.

Consumer agent. *CAs* submit consumers' service requests to *BAs*, which in turn contacts *SPAs* to acquire a set of resources. See Appendix 10, available in the online supplemental material.

4.2 Acquaintance Network and Service Capability Table

In [34], an agent's acquaintance network (*AN*) is a matrix in which the columns record the list of acquaintances (i.e., the list of agents that an agent knows), and the rows represent the service capabilities of the acquaintances. See Appendix 11, available in the online supplemental material.

Even though *SCTs* are reminiscent of the idea of *ANs*, they differ in terms of both the information stored and volatility. An *AN only* records the service capabilities of agents in the system. An *SCT* augments an *AN* by recording both: 1) the service capabilities of agents in the cloud system and 2) the states of cloud agents. A cloud agent can be in one of the following states: {"available," "unreachable," "failed," "busy"}. An agent is in the "available" (respectively, "unreachable") state when it responds (respectively, does not respond) to the request of another agent. If an agent has the service capabilities to satisfy the request of another agent but is unable to entertain the request (e.g., its server is down), then the agent is said to be in the "failed" state. An agent is in the "busy" state when it is executing a job request of some agent. In the agent-based cloud computing testbed, the states of cloud agents will change as they interact with other agents. Since the states of cloud agents can change frequently, the information stored in *SCTs* are more volatile than that in *ANs*. For instance, in *ANs*, the capability tables may be updated only when new agents join the system or when existing agents leave the system.

Consumer agents' SCTs. Each *CA* maintains an *SCT* of *BAs*. However, each *CA* only records the list of *BAs* (and their locations and states) that it knows but not the service capabilities of *BAs*. This is because *BAs* may accept potentially any service requests from *CAs*, then subcontract the service requests to other *BAs* or *SPAs*.

Broker agents' SCTs. A *BA* maintains two *SCTs*: 1) an *SCT* of other *BAs* together with their locations and their states, and 2) an *SCT* of *SPAs* together with their locations, service capabilities, and their states. Since a *BA* may accept service requests without being totally certain that it can enlist the services of *SPAs* with the required service capabilities, a *BA* may also enlist another *BA* for satisfying consumer agents' requirements. Hence, a *BA* needs to maintain *SCTs* of both *BAs* and *SPAs*.

Service provider agents' SCTs. An *SPA* maintains two *SCTs*: 1) an *SCT* of other *SPAs* and 2) an *SCT* of *RAs* under its administration. Both *SCTs* record the locations of agents, their service capabilities, and their states. When delegating service requests to its *RAs*, an *SPA* consults its *SCT* of *RAs*. The capabilities of an *SPA* are the aggregation of all the capabilities of the *RAs* under its administration.

Resource agents' SCTs. An *RA* maintains an *SCT* of other sibling *RAs* under the administration of the *SPA*. The *SCT* contains the locations of other *RAs*, their resource capabilities, and their states. If an *RA* is unable to fulfil a service requirement delegated by its *SPA*, it can subdelegate the service request to another sibling *RA*.

4.3 SCTs and Focused Selection Contract Net Protocol

All agents in the testbed (Fig. 10) adopt *FSCNP* which considerably augments and extends the classical *CNP* [21] for selecting and subcontracting cloud resources to satisfy consumers' service requirements. In *CNP*, agents have two roles: manager (client) or contractor (server). An agent requiring the services or resources of other agents plays the role of a manager and sends request messages or call-for-proposals to other agents. Agents playing the role of a contractor listen to call-for-proposals, evaluate the list of call-for-proposals, and submit bids for contracts. Managers evaluate the bids from contractors to provide the service, and award the contract to the most appropriate contractor based on its service capabilities.

FSCNP differs from *CNP* in the following ways by allowing agents to:

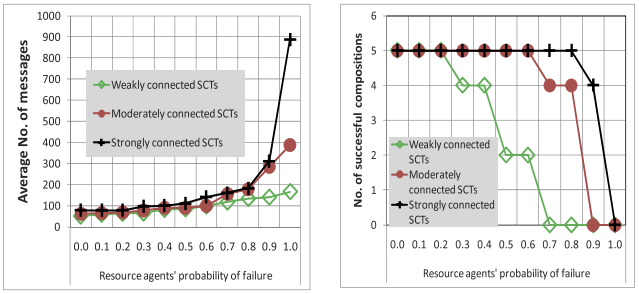
1. focus their service selections by interacting only with other agents that provide the relevant services,
2. assume multiple roles,
3. propagate and integrate the service results obtained from the multiple concurrent subcontracting interactions, and
4. react to failures by restarting the contracting process.

Focused selection. In the multiagent systems literature, *AN* and *CNP* are independent approaches of cooperative problem-solving [34]. In *CNP*, an agent attempts to select the services of other agents by broadcasting its requests to *all* other agents in the systems. In *FSCNP*, an agent consults its *SCT* to determine to which agents in the cloud system it should send its request message. By *only* sending messages to selected agents with relevant service capabilities, the interactions among cloud agents in *FSCNP* are more efficient because 1) the number of messages exchanged among cloud agents are considerably reduced, and 2) cloud agents only focus on interacting with a subset of agents in the cloud system that provides the relevant cloud services.

Playing multiple roles. In *FSCNP*, an agent can play the role of a contractor by accepting requests for performing a service to satisfy the set of requirements from a requesting cloud agent, and can concurrently play the roles of multiple managers to subcontract the service requests it received to other cloud agents. In *CNP*, an agent can only play the role of either a manager or a contractor at a time.

Integrating results and dealing with failures. Agents in *FSCNP* can propagate as well as integrate the service results obtained from the multiple concurrent subcontracting interactions. This is particularly essential to cloud service composition in this work because a *BA* needs to compose and combine a set of resources from multiple *SPAs*, and provides a single virtualized service to *CAs*. Furthermore, a cloud agent in *FSCNP* can react to failures by restarting a (sub)contracting process and keep track of its previous interactions by updating the states of other cloud agents recorded in its *SCTs*. In *CNP*, agents are not designed to combine results from multiple concurrent subcontracting interactions and they do not record the states of other agents.

Empirical results. To evaluate the self-organization capabilities of agents, a series of experiments was conducted in [35], [36] using the agent-based testbed in Section 4.1, and all the agents adopted *FSCNP* and *SCTs*. Existing approaches in multiagent systems using *AN* only specify that an acquaintance network is a partial representation of the capabilities of other agents but do *not* examine the effect of having different degrees of knowledge represented in an *AN*. This work studies the effect of having different degrees of knowledge represented in an *SCT* by conducting experiments to evaluate the performance of cloud agents when they are equipped with *SCTs* that contain different degrees of knowledge of the service capabilities of other agents. By having more (respectively, less) knowledge of the service capabilities of other agents in the cloud system, a cloud agent is said to be strongly (respectively, weakly) connected to other agents. Three sets of experiments were carried out using different connectivity degrees of *SCTs*: 1) weakly connected *SCTs* (having 1 to 33 percent knowledge of the service capabilities of other agents), 2) moderately connected *SCTs* (having 34 to 66 percent knowledge of the service capabilities of other agents), and 3) strongly connected *SCTs* (having 67 to 100 percent knowledge of the service capabilities of other agents). These experiments are designed to evaluate the agents' abilities to interact among themselves for autonomous (re)configuration and reselection of cloud services in response to failure of the resources. When *RAs* fail, the resources they manage cannot be accessed and there is a need for agents to reorganize among themselves to reconfigure a new set of resources to satisfy the service requirements. In the experiments, *RAs* were designed to fail with different failure probabilities of 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0. Two performance measures were used: 1) the number of successful service compositions and 2) the number of messages exchanged. Whereas measuring the number of successful service compositions for different failure probabilities of *RAs* provides a means to evaluate the effectiveness of using *FSCNP* and *SCTs* to automate cloud service composition, recording the number of messages exchanged among agents measures the efficiency of agent-based cloud service composition.



(a) (b)

Fig. 11. Empirical results of agent-based cloud service composition.

Figs. 11a and 11b show the number of successful compositions and the number of messages exchanged for agents adopting 1) weakly connected *SCTs*, 2) moderately connected *SCTs*, and 3) strongly connected *SCTs* under different failure probabilities of *SCTs*. From Figs. 11a and 11b, it can be observed that with an increase in the probability of failure of *RAs*, agents adopting strongly connected *SCTs* achieved the highest number of successful compositions but exchanged the largest number of messages. Agents adopting strongly connected *SCTs* have a large pool of acquaintances and are more likely to have higher chances of acquiring more cloud resources. Hence, they have a higher probability of successfully acquiring the required cloud resources to compose cloud services. However, since all agents follow *FSCNP*, agents adopting strongly connected *SCTs* exchange more messages in the process of service composition.

Analyzing the number of messages exchanged. The number of messages sent by an agent relies on the degree of knowledge of the service capabilities of other agents recorded in its *SCT*. An example to analyze the number of messages sent by a *BA* is as follows (due to space limitation, the analyses of other agents, e.g., *SPAs* and *RAs* will not be given here). When a *CA* submits a service request to the *BA*, the *BA* needs to subcontract the service requests to p *SPAs*, and p depends on the number of requirements submitted by the *CA*. For each subcontract, the *BA* consults its *SCT* to select q *SPAs* and send q call-for-proposal messages to these *SPAs*. For each subcontract, when the *BA* receives the proposal messages from the q *SPAs*, the *BA* will send out q response messages consisting of 1 accept-proposal to SPA_k and $q-1$ reject-proposals to $\{SPA_1, \ldots, SPA_{k-1}, SPA_{k+1}, \ldots, SPA_q\}$. Suppose that SPA_k suddenly enters the "failed" state, the failure is propagated to the *BA*. Then, the *BA* sends $q-1$ call-for-proposal messages to $\{SPA_1, \ldots, SPA_{k-1}, SPA_{k+1}, \ldots, SPA_q\}$, and in turn, the *BA* will receive $q-1$ responses. This process continues until the *BA* awards the subcontract to an *SPA* that is in the "available" state. Thus, in the worst case, the *BA* sends:

$$\begin{aligned} & p(2q + 2(q-1) + 2(q-2) + , \cdots, +2(2) + 2(1)) \\ & \quad = 2p(q + (q-1) + (q-2) + , \cdots, +(2) + (1)) \\ & \quad = 2p(q(q+1)/2) = p^* q(q+1) \end{aligned}$$

messages. In strongly (respectively, weakly) connected *SCTs*, q will be larger (respectively, smaller), and hence, more messages are exchanged as evidenced by Fig. 11b.

With a probability of 1.0 of entering the "failure" state, close to 900 messages were exchanged among agents with strongly connected *SCTs* and only less than 200 messages for agents with weakly connected *SCTs*. Furthermore, if *BAs* follow *CNP* [21], and if there are a total of t *SPAs*, and $q < t$, then in the worst case, the number of messages exchanged will be $p^*t(t+1)$, which can potentially be much larger than $p^*q(q+1)$. Hence, *FSCNP* is more efficient than *CNP* in terms of the number of messages exchanged.

5 Conclusion and Future Work

The novelty and significance of this work are that by introducing the idea of applying the agent paradigm to building software tools and testbeds for managing cloud resources, this work advances the state of the art in two ways. From the perspective of cloud computing, this work contributes to the field of cloud resource management by devising several novel approaches for facilitating cloud service discovery, service negotiation, and service composition. From the perspective of multiagent systems, this work demonstrates the application of 1) cooperative problem-solving paradigms to automating cloud service composition, 2) complex and concurrent negotiations to cloud commerce, and 3) software agents to building a cloud search engine.

The contributions of this work are detailed as follows:

1. *Cloudle*. An agent-based cloud service search engine was developed. To the best of the author's knowledge, *Cloudle* is the first search engine designed specifically for discovering cloud services. The novelty of *Cloudle* is that it is a multicriteria search engine that accepts as its inputs functional, technical, and budgetary requirements from consumers. Its *SDA* reasons about the similarities among cloud services and determines different levels of matching between the respective prices and schedules of a consumer and a provider. A cloud ontology and three novel types of service reasoning: similarity reasoning, compatibility reasoning, and numerical reasoning are devised. Multiple cloud crawlers maintain *Cloudle's* database by extracting relevant webpages from the *WWW*.
2. A complex cloud negotiation mechanism was devised to support cloud commerce. To the best of the author's knowledge, this complex negotiation mechanism is the earliest work to support concurrent negotiation activities in interrelated markets: a cloud service market between consumer agents and broker agents, and multiple cloud resource markets between broker agents and provider agents. In the multilateral negotiation in a cloud service market, empirical results show that consumer agents adopting the *BPE* and *MDA* strategies can better respond to different market conditions because they do not make excessive (respectively, inadequate) amounts of concessions in favorable (respectively, unfavorable) markets. Whereas empirical evidence in [30] *only* compared the behaviors of *MDA* and *BPE* in a *small number* of consumer-to-broker ratios, this work generalizes the results in [30] by providing a mathematical analysis of *MDA* and *BPE* in very extreme markets (e.g., when the consumer-to-broker ratio approaches infinity or zero). In the concurrent negotiations in multiple cloud resource markets, empirical results show that broker agents adopting *UOC* achieved significantly higher utilities, higher success rates, and faster speed than broker agents adopting *PCS*. By introducing the novel idea that negotiation activities are not restricted to only one market and concurrent negotiation activities can be carried out in multiple interrelated cloud markets, this work offers an entirely new branch of thinking in agent-based negotiations [37], [38].
3. To automate cloud service composition, the *FSCNP* was devised for specifying the interactions of cloud agents and *SCTs* were used to record the service capabilities of cloud agents. In *FSCNP*, agents focus on selecting relevant cloud services by consulting *SCTs*, thereby reducing the number of messages exchanged among cloud agents. Empirical results show that agents generally achieved higher success rates in composing cloud services, and agents adopting strongly connected *SCTs* achieved the highest number of successful compositions but exchanged the largest number of messages. An analysis on the number of messages exchanged among agents in *FSCNP* was also carried out to study the worst case.

This paper has reported *some* of the ongoing work in the Multiagent and Cloud Computing Systems Laboratory, led by the author. Reporting some of the earliest works by the author in adopting the agent paradigm for designing and constructing software tools and testbeds for cloud resource management, this paper has only taken the first step to show that agent-based problem-solving approaches and protocols provide effective methods for managing cloud resources. Some of the future work and challenges are given as follows:

1. In its present form, the cloud negotiation mechanism reported in Section 3 only considers negotiation of resource pricing but does not consider negotiation of other issues such as *QoS* and time slot negotiation. One of the unaddressed issues in cloud service negotiation is to devise a negotiation mechanism for cloud agents to reach agreements on resource price and *QoS*, as well as other parameters such as the time slot for scheduling of services.
2. Given that cloud services can be dynamically removed or added, creating and maintaining the *SCTs* of cloud agents can be a difficult problem. Devising some mechanisms to automatically record and maintain a cloud agent's list of acquaintances and their service capabilities is among the list of unaddressed issues in agent-based cloud service composition.
3. A more sophisticated service selection mechanism may be needed to deal with changing consumers' requirements. For example, owing to its changing job requirements, a consumer may request more storage capacity in addition to its current requests contracted to a broker.

It is hoped that this paper will shed new light in designing and constructing software tools for cloud resource management, and inspire others to take up the challenge of conducting research in agent-based cloud computing.

ACKNOWLEDGMENTS

The author would like to thank the Editor-in-chief, the Guest Editors, and the anonymous referees for their comments and suggestions. Part of the programming and experimentation of this project (KRF-2009-220-D00092) was carried out by J.O. Gutierrez-Garcia, J. Kang, D. Yoo, and S. So. This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MEST) (KRF-2009-220-D00092).

REFERENCES

[1] M. Miller, *Cloud Computing: Web-Based Applications that Change the Way You Work and Collaborate Online.* Que, 2009.
[2] I. Foster et al., "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. Grid Computing Environments Workshop (GCE '08),* pp. 1-10, Nov. 2008.
[3] R. Buyya et al., "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems,* vol. 25, no. 6, pp. 599-616, June 2009.
[4] K.M. Sim, "Agent-Based Cloud Commerce," *Proc. IEEE Int'l Conf. Industrial Eng. and Eng. Management,* pp. 717-721, 2009.
[5] M. Wooldridge, *An Introduction to Multiagent Systems,* second ed. John Wiley & Sons, 2009.
[6] K.M. Sim, "Towards Complex Negotiation for Cloud Economy," *Proc. Int'l Conf. Advances in Grid and Pervasive Computing (GPC '10),* R.S. Chang et al., eds., pp. 395-406, 2010.
[7] K.M. Sim, "Towards Agent-Based Cloud Markets (Position Paper)," *Proc. Int'l Conf. E-CASE, and E-Technology,* pp. 2571-2573, Jan. 2010.
[8] K.M. Sim, "Complex and Concurrent Negotiations for Multiple Interrelated E-Markets," *IEEE Trans. Systems, Man and Cybernetics, Part B,* preprint, 2012, doi:10.1109/TSMCB.2012.2204742.
[9] K.P. Joshi, T. Finin, and Y. Yesha, "Integrated Lifecycle of IT Services in a Cloud Environment," *Proc. Third Int'l Conf. Virtual Computing Initiative (ICVCI '09),* pp. 475-478, 2009.
[10] K.M. Sim and B. Shi, "Concurrent Negotiation and Coordination for Controlling Grid Resource Co-Allocation," *IEEE Trans. Systems, Man and Cybernetics, Part B,* vol. 40, no. 2, pp. 753-766, June 2010.
[11] K.M. Sim, "Grid Resource Negotiation: Survey and New Directions," *IEEE Trans. Systems, Man and Cybernetics, Part C,* vol. 40, no. 3, pp. 245-257, May 2010.
[12] K.M. Sim, "Evolving Fuzzy Rules for Relaxed-Criteria Negotiation," *IEEE Trans. Systems, Man and Cybernetics, Part B,* vol. 38, no. 6, pp. 1486-1500, Dec. 2008.
[13] K.M. Sim and B. Shi, "Adaptive Commitment Management Strategy Profiles for Concurrent Negotiations," *Proc. First Int'l Workshop Agent-Based Complex Automated Negotiations (ACAN) held in Conjunction with Seventh Int'l Conf. Autonomous Agents and Multi-Agent Systems (AAMAS),* pp. 16-23, 2008.
[14] K.M. Sim, "G-Commerce, Market-Driven G-Negotiation Agents and Grid Resource Management," *IEEE Trans. Systems, Man and Cybernetics, Part B,* vol. 36, no. 6, pp. 1381-1394, Dec. 2006.
[15] K.M. Sim, "A Survey of Bargaining Models for Grid Resource Allocation," *ACM SIGeCOM: E-Commerce Exchanges,* vol. 5, no. 5, pp. 22-32, Dec. 2005.
[16] K.M. Sim, "Equilibria, Prudent Compromises, and the 'Waiting' Game," *IEEE Trans. System, Man, Cybernetics B,* vol. 35, no. 4, pp. 712-724, Aug. 2005.
[17] K.M. Sim and S.Y. Wang, "Flexible Negotiation Agent with Relaxed Decision Rules," *IEEE Trans. Systems, Man and Cybernetics, Part B,* vol. 34, no. 3, pp. 1602-1608, June 2004.
[18] K.M. Sim, "Negotiation Agents that Make Prudent Compromises and Are Slightly Flexible in Reaching Consensus," *Computational Intelligence,* vol. 20, no. 4, pp. 643-662, 2004.
[19] K.M. Sim and C.Y. Choi, "Agents that React to Changing Market Situations," *IEEE Trans. System, Man, Cybernetics B,* vol. 33, no. 2, pp. 188-201, Apr. 2003.
[20] K.M. Sim, "A Market-Driven Model for Designing Negotiation Agents," *Computational Intelligence,* vol. 18, no. 4, pp. 618-637, 2002.
[21] R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Trans. Computers,* vol. C-29, no. 12, pp. 1104-1113, Dec. 1980.
[22] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages," *Artificial Intelligence,* vol. 8, no. 3, pp. 323-364, 1977.
[23] J. Kang and K.M. Sim, "*Cloudle*: A Multi-Criteria Cloud Service Search Engine," *Proc. IEEE Asia-Pacific Services Computing Conf.,* Dec. 2010.
[24] J. Kang and K.M. Sim, "*Cloudle*: An Ontology-Enhanced Cloud Service Search Engine," *Proc. First Int'l Workshop Cloud Information System Eng., Collocated with 11th Int'l Conf. Web Information System Eng.,* Dec. 2010.
[25] J. Kang and K.M. Sim, "*Cloudle*: An Agent-Based Cloud Search Engine that Consults a Cloud Ontology," *Proc. Int'l Conf. Cloud Computing and Virtualization,* pp. 312-318, May 2010.
[26] L. Youseff, M. Butrico, and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," *Proc. Grid Computing Environments Workshop (GCE '08),* pp. 1-10, 2008.
[27] T. Andreasen, H. Bulskov, and R. Kanppe, "From Ontology over Similarity to Query Evaluation," *Proc. Second Int'l Conf. Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems (ODBASE),* Nov. 2003.
[28] L. Han and D. Berry, "Semantic-Supported and Agent-Based Decentralized Grid Resource Discovery," *Future Generation Computer Systems,* vol. 24, no. 4, pp. 806-812, Apr. 2008.
[29] A. Rubinstein, "Perfect Equilibrium in a Bargaining Model," *Econometrica,* vol. 50, no. 1, pp. 97-109, Jan. 1982.
[30] D. Yoo and K.M. Sim, "A Multilateral Negotiation Model for Cloud Service Market," *Proc. Conf. Grid and Distributed Computing,* Dec. 2010.
[31] T. Sandholm and V. Lesser, "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework," *Reading in Agents,* pp. 66-73, Morgan Kaufmann, Jan. 1997.
[32] T.D. Nguyen and N.R. Jennings, "Managing Commitments in Multiple Concurrent Negotiations," *Int'l J. Electronic Commerce Research and Applications,* vol. 4, no. 4, pp. 362-376, 2005.
[33] I. Rahwan et al., "Intelligent Agents for Automated One-to-Many E-Commerce Negotiation," *Proc. 25th Australasian Conf. Computer Science,* vol. 4, pp. 197-204, 2002.
[34] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison Wesley, 1999.
[35] J.O. Gutierrez-Garcia and K.M. Sim, "Self-Organizing Agents for Service Composition in Cloud Computing," *Proc. Second IEEE Int'l Conf. Cloud Computing Technology and Science,* 2010.
[36] J.O. Gutierrez-Garcia and K.M. Sim, "Agent-Based Service Composition in Cloud Computing," *Proc. 2010 Conf. Grid and Distributed Computing,* Dec. 2010.
[37] J. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.* MIT Press, 1994.
[38] N.R. Jennings et al., "Automated Negotiation: Prospects, Methods, and Challenges," *Int'l J. Group Decision Negotiation,* vol. 10, no. 2, pp. 199-215, 2001.

Kwang Mong Sim received the BSc (Hon) degree summa cum laude from the University of Ottawa, Ontario, Canada, and the MSc and PhD degrees from the University of Calgary, Alberta, Canada. He is an associate editor for the *IEEE Transactions on Systems, Man and Cybernetics-Part C* and the *International Journal of Cloud Computing*, and is an editorial/advisory board member of many international journals. He has also been the guest editor of five IEEE journal special issues in agent-based grid computing and automated negotiation, including a special issue on grid resource management in the *IEEE Systems Journal*. He is the Medway Chair and Professor of Computer Science at the University of Kent, Chatham Maritime, Kent, United Kingdom, and was the director of the Multiagent and Cloud Computing Laboratory, Gwangju Institute of Science and Technology, South Korea. He has delivered many keynote lectures on agent-based cloud computing and automated negotiation in many international conferences, and has published many survey papers, technical papers, and award-winning conference papers in cloud/grid computing and multiagent systems. He served as a referee for the US National Science Foundation. He is a senior member of the IEEE.