Agent-based Approaches for Intelligent Intercloud Resource Allocation

Kwang Mong Sim  Fellow, IET, Senior Member, IEEE

Abstract—Whereas an Intercloud is an interconnected global "cloud of clouds" that enables each cloud to tap into resources of other clouds, interactions among Intercloud stakeholders are complex because Intercloud resources are distributed and controlled by different clouds. "Agent-based cloud computing" involves the construction of agents for bolstering discovery, matching, selection, composition, negotiation, scheduling, workflow, and monitoring of Intercloud resources. An agent is a computer system that is capable of making decisions independently and interacting with other agents through cooperation, coordination, and negotiation. Using an agent-based approach, characteristics associated with intelligent behaviors of agents such as interacting socially through cooperation, coordination, and negotiation can be built into clouds. This paper 1) discusses the significance and advantages of using an agent paradigm for Intercloud resource allocation, 2) reviews representative models of agent-based Intercloud resource allocation and provides a comparison among these models, 3) compares agent-based and non-agent-based approaches for task executions in multiple clouds, and 4) provides pointers to future directions.

Index Terms— Autonomous agents, cloud computing, cooperative problem solving, distributed artificial intelligence, intelligent agents, Intercloud, resource management.

1. INTRODUCTION

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [1]. One of the essential aspects of cloud computing is creating the illusion that “infinite” computing resources are available on demand [2]. However, the resources held by a single cloud are usually limited and it may not be able to deal with a sudden surge in user demands (see p.3 [3]; p.16, [19]). An Intercloud [3] is an interconnected global “cloud of clouds” that enables cooperation among clouds. In an Intercloud, each cloud can tap into resources of other clouds when it does not have sufficient resources to satisfy consumers’ requests.

Interclouds are classified into federated clouds and multi-clouds [4]. In a federated cloud [5], providers voluntarily interconnect their infrastructures to enable sharing and exchange of resources among themselves. Federated clouds are classified into centralized (resource allocation performed by a central entity) and peer-to-peer (no central authority) modes [4]. Clouds interconnected at the same layer (e.g., between two or more IaaS providers) is called a horizontal federation and clouds interconnected at different layers (e.g., between a PaaS provider and an IaaS provider) is called a vertical federation [6] (see Appendix A in supplemental material). In a multi-cloud, cloud providers do not necessarily volunteer to interconnect and share their infrastructures, and consumers are responsible for managing resources across multiple clouds. In a broker aggregated service multi-cloud [4], cloud brokers [7] provide resource selection and aggregation services.

An agent is a computer system that is capable of making decisions independently, carrying out actions autonomously, and interacting with other agents through cooperation (working together and drawing on each other’s knowledge and capabilities), coordination (achieving the state in which their actions fit in well with others), and negotiation (trying to reach agreements on some matters) [8].

In 2009, Sim [9] published the earliest work documenting the idea of using intelligent agents for bolstering cloud resource trading. The emerging area of “Agent-based cloud computing” (ABCC) includes a body of works involving the construction of agents for bolstering discovery [10,23], matching [6], selection [26,29], composition [10,30], negotiation [10,11,35,36,42,44,45], scheduling [47,48], workflow [53], and monitoring [31-33] of Intercloud resources.

Motivation of ABCC: Intercloud resource allocation is a complex problem [12,13]. In an Intercloud, (heterogeneous) computing resources owned and administered by different cloud providers are pooled to serve multiple consumers. Since Intercloud resources are distributed and are controlled and managed by different clouds, interactions among clouds are complex [12, p.3]. Additionally, even though clouds join an Intercloud by granting access to their resources to other clouds, it is essential to ensure that they continue to be self-governing entities with a certain degree of autonomy [12, p.3]. These issues motivate the need for developing agents to 1) automate complex interactions among components in an Intercloud and 2) enable Intercloud components to operate autonomously (see Appendix A). A review and comparison of representative agent-based approaches for bolstering Intercloud resource allocation are given in sections 2 and 3, respectively.

Advantages of ABCC: ABCC advances the state of the art in two ways. From the perspective of agent-based computing, ABCC contributes new agent-based methods (e.g., new complex negotiation mechanisms [45] and new cooperative problem-solving techniques [10,30]) and demonstrates their applications to Intercloud resource allocation. From the perspective of cloud computing, ABCC contributes novel resource management approaches for facilitating Intercloud resource discovery, matching, selection, composition, negotiation, scheduling, workflow, and monitoring. The
advantages of agent-based approaches over non-agent-based approaches for task executions in multiple clouds are discussed in section 4. 

**Significance of ABCC**: Adopting an agent paradigm enables clouds to maintain their autonomy and interact more intelligently and more efficiently through social interactions, and allows agent-based Intercloud resource allocation systems to be designed with desirable properties specified and proven using game theory.

1) Modeling an Intercloud as a multiagent system (MAS) enables individual clouds to operate as autonomous components within a larger interconnected system. Using agents to automate the interactions among clouds allows each cloud to have more control over its own resources by having more flexibility to implement its own scheduling policies (e.g., scheduling its own tasks at some preferred time slots) while committing to execute the tasks of others (see sections 4.2 and 4.3). In an MAS, conflicting schedules may be resolved through automated negotiation among agents (section 2.4).

2) By modeling an Intercloud as a multiagent system, “intelligent” characteristics can be built into clouds. “Cloud intelligence” [14] refers to the characteristics of cloud agents that are associated with intelligent behaviors of agents. In ABCC, agents are designed to automatically establish service contracts through negotiation, integrate multiple resources from different clouds into a unified service through cooperation, and manage concurrent workflow and schedule parallel execution of tasks in multiple clouds through coordination (section 2). The ability to interact socially through cooperation, coordination, and negotiation is considered as an intelligent characteristic of agents [8, p.26].

3) Since agents’ interactions can be analyzed using game theory [15,16], by modeling an Intercloud as a multiagent system, the interaction protocols (i.e., rules that govern the interactions) and strategies (specifications of what to do in every alternative during an interaction) of cloud agents can be designed based on well-known solution concepts from game theory [17,18]. Game-theoretic solution concepts provide the guiding principles for designers to specify and prove desirable properties of agent-based cloud computing systems (e.g., optimality, stability, and fairness) [13].

**Scope of this survey**: Even though a wide range of issues is involved when constructing an Intercloud, e.g., connectivity [19], interoperability [19,20], security [21], communication [22] and others [19,20], this survey only focuses on reviewing and comparing agent-based approaches that specifically address the Intercloud resource allocation problem. Works addressing issues other than Intercloud resource allocation (e.g., interoperability and security) are outside the scope of this survey.

## 2. Agent-based Intercloud Models

This section reviews representative models on agent-based approaches for the discovery and matching (section 2.1), selection and composition (section 2.2), monitoring (section 2.3), negotiation (section 2.4), and scheduling and workflow (section 2.5) of Intercloud resources.

### 2.1 Discovery and Matching

This sub-section reviews two approaches for finding appropriate resources that comply with service requests.

1) **Matching Requests to Services using Multiple Brokers**: Kang and Sim [23] developed an agent-based multi-cloud testbed for service discovery consisting of consumer agents (CAs), broker agents (BAs), and provider agents (PAs). Each CA (respectively, PA) sends its requests (respectively, advertisements) to a BA. Using similarity reasoning, each BA attempts to match requests to advertisements (similar to allocation by brokers [24, pp. 343-345]) (See Appendix B in supplemental material). Similarity reasoning determines the degree of similarity between concepts by counting their common properties. In matching prices and time slots of PAs and CAs, a BA attempts to match CAs with PAs that 1) can accommodate the time slots specified by CAs and 2) have acceptable prices that do not deviate too much from those of CAs. Multiple BAs are used in the testbed. To enhance the chance of finding a good match, each BA can transfer unmatched requests from CAs or unmatched advertisements from PAs to other BAs for further matching. Even though empirical results in [23] suggest that this approach can match requests to advertisements with reasonably high success rates, the disadvantage is that an additional set of special agents (BAs) is needed to manage the matching process by centralizing the requests and advertisements.

2) **Agent-based Cross-cloud Federation**: Celesti et al. [6] developed an agent-based testbed for horizontal Intercloud federation. The testbed consists of: home clouds (HCS) and foreign clouds (FCs). HCs are clouds that require additional storage or computing capacities from other clouds. FCs lease part of their computing and storage capacities to HCs. A cloud provider can simultaneously assume both the roles of an HC and an FC. In [6], each cloud has a cross-cloud federation manager (CCFM) that interacts with other CCFMs in a three-phase process (discovery, match-making, and authentication) using three types of agents: discovery agent, match-making agent and authentication agent.

**Discovery agent (DA)**: The Intercloud discovery process in [6] is managed by DAs. DAs communicate among themselves using the publish-subscribe messaging pattern [25] where senders of messages are called publishers and receivers are called subscribers. Each (publisher) DA publishes information about the states and service capabilities of the resources of the cloud that it represents at a centralized location (which is an intermediary message broker or event bus [25]). A set of authorized (subscriber) DAs representing other clouds can access the information in the centralized location. In the publish-subscribe messaging pattern, published messages are characterized into classes, without knowledge of the subscribers. Similarly, by subscribing to one or more message classes, subscribers only receive messages that are of interest without knowledge of the publishers. During the discovery process, the DA of each HC can compile a list of potential FCs by retrieving the information about their service capabilities and availability from the centralized location.
Match-making agent (MA): In forming a cloud federation, MAs select appropriate FCs from the list of discovered clouds by attempting to align requirements of HCs with policies of FCs. Requirements of HCs and policies of FCs are represented as sets of rules, which respectively describe the required resources and resources offered based on some conditions. An example of a rule specifying the requirement of an HC given in [6] is “I accept CPU X1, RAM Y1, with QoS Z1: from any foreign cloud trusted with the identity provider $T^*$. An example of a rule specifying the policy of an FC is given as follows: “I provide CPU X1, RAM Y1, with QoS Z1: to all clouds except cloud A and I’m trusted with the identity providers $T, H^*$ [6].

The match-making process performed by the MA of an HC consists of two stages.

1) In the pre-filtering stage, the MA analyzes both quantifiable information (e.g., the amount of RAM that is available can be expressed as a numeric value) and unquantifiable information (e.g., trusted identity providers (IdPS) and the list of black-listed clouds are expressed as strings).

2) Representing the set of quantifiable parameters of each cloud (both HC and FC) as an N dimensional vector, the MA selects the most appropriate FC(s) by comparing each of the quantifiable parameters of both the HC and each FC. The degree of match between a quantifiable parameter of an HC and the corresponding parameter of an FC is estimated by computing the Euclidian distance between both parameters.

Authentication agent (AA): At the start of the authentication phase, the AA of an HC initiates an exchange of authentication information with the AAs of FCs. The exchange of meta-data involves trusted third parties (IdPS). The authentication process is based on the concept of single sign-on (SSO) authentication, where an HC can gain access to the resources provided by the FCs without further identity checks if both the HC and FCs have established a trust context that allows cross-cloud resource provisioning. The MA of each HC authenticates itself with the FCs using a digital identity that is issued by a trusted IdP.

Since the selection of resources in [6] is based only on 1) the matching features between resource specifications and 2) trusted IdPS, there is no economic mechanism for pricing resources and optimizing resource utilization in a cloud federation. This issue was investigated in [13].

2.2 Selection and Composition

This subsection reviews three approaches for selecting and combining service components from different cloud providers.

1) CNP for Resource Selection: The agent-based multi-cloud resource selection testbed developed by Ejarque et al. [26] consists of job agents (JAs) and resource agents (RAs). JAs send requests for resources on behalf of consumers. RAs interact with JAs and schedule resources for running jobs on behalf of providers. JAs and RAs interact using the Contract Net Protocol (CNP) [27,28] — a three-stage protocol specifying the interactions between manager agents (requesters of resources) and contractor agents (providers of resources) (see Appendix B). In stage 1, a JA (manager) attempts to select an appropriate resource by broadcasting a call for proposals (CFP) to execute a job to all RAs. In stage 2, each RA (contractor) evaluates the CFP and the JA’s requirements. RAs that fulfill the requirements reply to the JA with a proposal for executing the job. In stage 3, the JA selects the RA with the best proposal (e.g., the lowest price) for executing the job and rejects all other proposals.

2) CNP with Mediator for Resource Selection: “Cloud Agency” developed by Venticinque et al. [29] is another agent-based multi-cloud testbed that adopts the CNP for selecting the best cloud resources for consumers. In “Cloud Agency”, client agents (cLAs) and vendor agents (VAs) act on behalf of consumers and providers, respectively. The interactions between cLAs and VAs are mediated by mediator agents (MeAs) and broker instances (BIs). On receiving a CFP from a cLA, an MeA creates a BI to handle the CFP. The BI searches for available providers and solicits proposals from VAs. Then, the BI selects the best proposal from the set of proposals from available VAs and notifies the cLA. Upon receiving a confirmation from the cLA, the BI allocates the resource to the client.

3) Focused-selection CNP for Service Composition: Gutierrez-Garcia and Sim [30] developed an agent-based multi-cloud testbed for service composition. The testbed consists of consumer agents (CAs), broker agents (BAs), provider agents (PAs), and resource agents (RAs). Each CA submits service requests to BAs and selects the BA with the best proposal. For each request from a CA, a BA selects a set of resources from a group of PAs, combines them into a single unified service, and sub-leases it to the CA. Each PA allocates and schedules resources, and coordinates the execution of jobs by interacting with a set of RAs that it manages. An RA controls a resource and returns job execution outputs. To integrate a set of resources from multiple cloud providers, agents in [30] adopt a novel CNP called the focused-selection contract net protocol (FSCNP) [11]. Instead of broadcasting a CFP to contractor agents, manager agents in FSCNP consult a service capability table (SCT) [11,30] to determine to which contractor agents they should send their service requests.

SCTs vs. ANs: As noted in [11], although SCTs are reminiscent of acquaintance networks (ANs) [24, pp. 348-349], they differ in terms of both the information stored and volatility. An AN is a “skills table” that only records the locations as well as the service and resource capabilities of (other) agents in the system (see Appendix B). An SCT augments an AN by recording: 1) the locations as well as the service and resource capabilities of agents in a cloud system and 2) the states of cloud agents. In general, a cloud agent can be in one of the following four states: (“available,” “unreachable,” “failed,” “busy”) [11, 30]. An agent is in the “available” (respectively, “unreachable”) state when it responds (respectively, does not respond) to the requests of other agents. If an agent has the resource capabilities to satisfy the requests of other agents but is not able to
entertain the requests due to some reason (e.g., its server is down), then the agent is in the “failed” state. An agent is in the “busy” state when its resources are tied up, i.e., it is executing jobs from other agents. Since the states of cloud agents can change very frequently, the information stored in SCTs is generally more volatile than that in ANs. In contrast, the skills tables in ANs may be updated only when new agents join or existing agents leave the system.

Consumer agents’ SCTs: Every CA maintains an SCT of BAs [11,30]. BAs can potentially accept any service requests from CAs, and subcontract the service requests to PAs or other BAs. Hence, a CA’s SCT only needs to record a list of BAs that it is acquainted with and their locations and states, and CAs need not record the service capabilities of BAs.

Broker agents’ SCTs: A BA maintains two SCTs: 1) an SCT of PAs together with their locations, resource and service capabilities, and their states, and 2) an SCT of other BAs together with their locations and states [11,30]. Each BA maintains SCTs of both BAs and PAs because in some situations when it is unable to enlist the services of a group of PAs with the required service capabilities, the BA may also enlist another BA for satisfying a CA’s requirements.

Service provider agents’ SCTs: A PA maintains two SCTs: 1) an SCT of RAs that it manages and 2) an SCT of other PAs [11,30]. Both SCTs record the locations of agents, their resource and service capabilities, and their states. To delegate service requests to its RAs, a PA consults its SCT of RAs. The service capabilities of a PA consist of the aggregated resource capabilities of all the RAs that it manages. However, a PA may also subcontract services to other PAs when 1) some of its RAs is in the “failed” state and/or 2) external services (e.g., cryptographic services) from other PAs are needed to satisfy a given job requirement.

Resource agents’ SCTs: Each RA maintains an SCT of other sibling RAs managed by the same PA [11,30]. An RA’s SCT contains the locations of other RAs, their resource capabilities, and their states. In case an RA is not able to fulfill a service requirement delegated by its PA, it can sub-delegate the service request to another sibling RA.

Agents’ interactions: CAs interact with BAs using the FSCNP. Each CA (manager) first consults its SCT of BAs, then selectively sends service requests to a group of BAs (contractors). By selecting BAs based on the information recorded in its SCT, each CA only needs to focus on its interactions with BAs that are available to provide services.

Similarly, BAs and PAs interact using the FSCNP. Upon receiving a request from a CA, a BA plays the role of a manager. To satisfy the service requirements of a CA, a BA may need to select a set of different resources from several groups of PAs and integrate them into a unified service. For each of the required resources, the BA consults its SCT of PAs then selectively sends a call for proposals to a group of PAs (contractors). The interactions between PAs and RAs, among BAs, among PAs, and among RAs are carried out in a similar manner: the manager agent 1) consults the appropriate SCT then 2) selectively sends requests to an appropriate group of contractor agents.

2.3 Monitoring

Cloud resource monitoring is a continuous activity used to gather information that assists and supports agents in making resource (re)-allocation decisions in the hope of optimizing resource utilization [31,32]. Endo et al. [31] classified cloud resource monitoring into: passive and active.

1) Passive Monitoring: Resource monitoring is passive when there are one or more agents (central managers [32]) collecting information from agents representing resources (workers [32]). Central managers gather information about the states of resources by sending polling messages to workers either continuously or on an on-demand basis.

Wei and Blake [33] developed an agent-based testbed consisting of extractors agents (EAs) and aggregator agents (AgAs) for monitoring resources in a multi-cloud. EAs are responsible for processing services of virtual machines (VMs), building service models by scanning the service repositories of VMs, and communicating with AgAs to register cloud resources. AgAs construct a cloud resources database by gathering different service models across different VMs. AgAs in different clouds communicate with each other to 1) exchange cloud resource information and 2) utilize the resources from other clouds if their consumer service requirements cannot be fulfilled by their own clouds. To facilitate resource discovery, AgAs adopt a check period relaxation protocol (CPRP) to monitor the states of cloud resources. In CPRP, AgAs send messages to EAs to check the states of the services in VMs. In this regard, AgAs and EAs correspond to central managers and workers in passive resource monitoring, respectively. Each time when an EA responds positively to an AgA, the AgA doubles the time interval for sending another check message to the EA. The doubling of the check period continues as long as an EA responds positively, but there is a threshold (set at 32 seconds by default) for the check period interval. When the threshold is reached, the protocol enters a “cautious relax” (CR) stage, such that if an EA responds positively to an AgA, the time interval for sending another check message is only increased by one second. The CR stage is designed to reduce prolonged delay in detecting the sudden discontinuation or failure of consistently well-performing cloud services.

2) Active Monitoring: Endo et al. [31,32] proposed the idea of active resource monitoring. In [32], workers are modeled as autonomic cloud agents (ACAs). Attached to cloud resources, ACAs are designed to monitor the states of the resources and are endowed with the capability to decide when to proactively send resource state information to central managers, modeled as autonomic cloud managers (ACMs). However, [32] did not specify how frequent ACs will update ACMs about the states of their resources.

2.4 Negotiation

In [34], the negotiation patterns for service level agreement (SLA) negotiation were classified into: 1) propose/accept
(“one-shot”), 2) propose/counter-propose*accept (“alternating”), and 3) .../solicit/accept. The “.../solicit/accept” pattern can be either “one-shot” or “alternating”, and the revocation of an acceptance is allowed. Hence, this pattern can be further classified into “one-shot revocable” and “alternating revocable”. Agent-based cloud negotiation models using these negotiation patterns are summarized as follows.

1) One-shot Cloud SLA Negotiation: The SLA negotiation in “Cloud Agency” [35] adopts the propose/accept pattern. SLA negotiation in “Cloud Agency” involves the interactions among client agents (cLAs), mediator agents (MeAs), a registry agent (ReA), an archival agent (ArA), and vendor agents (VAs). Whereas an ReA enables VAs to publish their services and MeAs to search for available services, an ArA stores the historical data of the quality of service (QoS) and resources offered by providers. SLA negotiation between cLAs and VAs is mediated by MeAs. Upon receiving a resource request from a cLA, an MeA retrieves a list of available resources from the ReA and sends a CFP to the corresponding VAs. When the MeA receives proposals from VAs, it evaluates each proposal based on the price and availability of the resource, the QoS guaranteed by the provider, and the statistics of the past performance of the provider obtained by requesting historical data from the ArA. Finally, the MeA awards the contract to the VA with the best proposal in terms of price, availability, QoS, and past performance, then notifies the cLA.

2) Price and Time-slot Alternating Negotiation: Adopting the propose/counter-propose*/accept pattern for SLA negotiation, Son and Sim [36] devised a price-and-time-slot negotiation (PTN) mechanism for enabling providers and customers to specify price and time slot preferences and potentially reach agreements on mutually acceptable prices and time slots. In [36], consumer agents (CAs) and provider agents (PAs) interact using the alternating offers protocol [37] (see Appendix C in supplemental material) by iteratively exchanging proposals to negotiate for mutually acceptable prices and time slots by making both concessions (e.g., compromising in price) and tradeoffs (e.g., settling with a less preferred time slot and paying a lower price). Both CAs and PAs adopt three classes of time-dependent concession-making strategies: (i) conditional (making smaller concessions in early rounds and larger concessions in later rounds), (ii) conciliatory (making larger concessions in early rounds and smaller concessions in later rounds), and (iii) linear (making a constant rate of concession) [38-40] (see Appendix C). The novelty of [36] is adopting a new tradeoff algorithm called the burst mode in which an agent is allowed to concurrently make multiple proposais, with each proposal consisting of a different pair of price and time slot that generates the same level of satisfaction. In the burst mode, an agent can provide more options for its trading partner without having to make concessions that lead to a lower level of satisfaction. For example, a PA can simultaneously send two proposals to a CA: 1) one proposal at a time slot when fewer of its resources are expected to be occupied but at a lower price and 2) the other at a time slot when many of its resources are occupied but at a higher price. A case study in [36] compares the PTN mechanism with Amazon’s spot instance pricing model [41] and shows that the PTN mechanism enables consumers to both save cost and utilize cloud services without interruption.

3) Price and Availability Alternating Negotiation: Dastjerdi et al. [42] developed cloud agents that enable consumers and providers to negotiate both resource price and availability. Even though it was not explicitly mentioned, agents in [42] adopt the propose/counter-propose*/accept negotiation pattern similar to the alternating offers protocol. Similar to [36], agents in [42] adopt time-dependent concession-making strategies. Two major classes of time-dependent concession-making strategies based on [43] are used: 1) Bouleware (conceding slowly initially, but making increasingly larger concessions as the deadline approaches) and 2) Conceder (making most of the concessions in early rounds, but does not concede much as negotiation progresses) (see Appendix C). In [42], an agent is designed to concede more on a less important issue (e.g., conceding more on price if availability is more important). To balance the utilization of resources, provider agents in [42] were designed to take into consideration the level of resource utilization when pricing their resources, i.e., conceding more (respectively, less) for resources that are less (respectively, more) utilized. However, agents in [42] were not designed to make tradeoffs between price and availability.

4) Three-tier Alternating Cloud Negotiation: Siebenhaar et al. [44] proposed a three-tier cloud SLA negotiation model comprising: 1) user tier (consisting of consumers and brokers (represented by consumer agents (CAs) and broker agents (BAs), respectively), 2) service tier (consisting of service providers represented by service provider agents (SAs)) and 3) resource tier (consisting of resource providers represented by resource agents (RA)). However, consumers do not participate in negotiation but delegate negotiation activities to BAs. Upon receiving a service composition request from a CA, a BA determines an appropriate set of cloud providers for each of the service requirements of the CA. In [44], negotiation activities were only carried out 1) between BAs and SAs and 2) between SAs and RAs. Each agent creates a coordination entity (CE) which in turn creates several negotiation entities (NE) to carry out negotiation with other agents. Using a two-phase protocol consisting of the warm-up phase and countdown phase, NEs adopt the propose/counter-propose*/accept pattern for SLA negotiation. In the warm-up phase, NEs of BAs (respectively, SAs) exchange pre-proposals and counter-pre-proposals in alternate rounds with NEs of SAs (respectively, RAs) until there is at least one acceptable pre-proposal. An NE of a BA (respectively, SA) selects the best pre-proposal among the acceptable pre-proposals of NEs of SAs (respectively, RAs), then sends a “temporary accept” message to the NE of the SA (respectively, RA) with the best pre-proposal and a “temporary reject” message to
NEs of all other SAs (respectively, RAs). In the countdown phase, the NE of the SA (respectively, RA) that receives a "temporary accept" message will send its definitive proposal while NEs of other SAs (respectively, RAs) that receive a "temporary reject" message in the warm-up phase can still revise and send their proposals in the hope of out-bidding the best pre-proposal. The countdown phase continues until the NE of the BA (respectively, SA) accepts the best "definitive" proposal from the NE of an SA (respectively, RA). NEs adopt three classes of time-dependent concession-making strategies based on [43]: 1) Boulevarde, 2) Linear, and 3) Conceder. However, even though NEs are designed to negotiate on three issues: price, availability, and response time, there is no mechanism for making tradeoffs among these three issues.

5) Concurrent Alternating Revocable Cloud Negotiations: Sim [10,45] devised a concurrent negotiation mechanism for bolstering parallel negotiation activities in the cloud market model proposed by Buyya et al. [46]. In [10,45], an agent-based testbed consisting of consumer agents (CAs), provider agents (PAs), and broker agents (BAs) was used to simulate the cloud market model in [46]. BAs accept service requests from CAs, purchase resources from PAs, dynamically compose collections of resources into bundled services to satisfy CAs’ requirements, then sub-lease the unified services to CAs. BAs carry out parallel negotiations with both 1) CAs for mutually acceptable terms for service provisions and 2) PAs for mutually acceptable terms for purchasing resources.

Since each CA can submit service requests to multiple BAs and each BA can accept requests from multiple CAs, a many-to-many negotiation model is used for modeling the negotiation between CAs and BAs. Adopting the alternating offers protocol in the many-to-many negotiation model, both CAs and BAs use the propose/counter-propose/accept pattern for SLA negotiation. Both CAs and BAs adopt the bargaining position estimation (BPE) strategy [45] for making concessions (see Appendix C). CAs and BAs respond to different resource market conditions by 1) observing concession patterns of their trading partners to estimate their bargaining positions and 2) making adjustable amounts of concession. Each CA (respectively, BA) estimates its bargaining position (BP) by considering 1) the amount of concession made in the current round by each BA (respectively, CA) relative to 2) the average amount of concessions made in the previous rounds by each BA (respectively, CA). An agent is likely in a relatively strong (respectively, weak) BP if its trading partners are making relatively larger (respectively, smaller) amounts of concessions. An agent with a stronger (respectively, weaker) BP will make a smaller (respectively, larger) concession.

In [10,45], a concurrent one-to-many negotiation mechanism was devised to model the concurrent negotiation activities between each BA and different groups of PAs. This is because a cloud service may be dynamically composed using multiple types of resources. Hence, each BA potentially needs to negotiate with multiple groups of cloud providers in multiple types of cloud resource markets. The concurrent cloud negotiation mechanism consists of 1) a coordinator module and 2) a set of commitment managers \(CM_1, \ldots, CM_n\). The coordinator module coordinates the parallel negotiation activities for acquiring \(n\) different types of cloud resources in \(n\) different resource markets. In each cloud resource market, a BA negotiates simultaneously with multiple PAs for one type of cloud resource. Adopting the alternating revocable pattern for SLA negotiation, both BAs and PAs can be freed from a contract by paying penalty fees to their trading partners (i.e., each agent can decommit (break) a contract). In negotiating for each type of cloud resource in a resource market, a BA has a commitment manager (CM) that manages both commitments and decommitments of (intermediate) contracts.

Two major algorithms of the concurrent cloud negotiation mechanism include: 1) an algorithm for establishing contracts and managing (de-)commitments of contracts and 2) an algorithm for coordinating the parallel negotiation activities.

The contracting algorithm [45] generally consists of the following. At each negotiation round, the CM determines whether to accept a proposal from a PA, or whether to break an existing contract and take up a new and more favorable contract by paying a penalty fee. To determine if a PA’s proposal is acceptable, the CM first computes the probability that the PA will break a contract (details are given in [45]). Since a resource can be requested by multiple BAs simultaneously, a PA can potentially break an intermediate contract. A PA’s proposal is acceptable to the BA if the PA’s proposed price equals to or is below the BA’s proposed price. If there is more than one acceptable proposal from PAs, the CM accepts the proposal that generates the highest utility for the BA (i.e., the proposal with the lowest price). In case the BA has previously reached an intermediate contract with another PA, then it will first decommit the contract by paying a penalty fee before it accepts a new and more favorable proposal. If none of the proposals from PAs is acceptable, the CM revises its proposal by making concession using the BPE strategy.

The coordination algorithm [45] generally consists of: 1) predicting the change in the expected outcome in each one-to-many negotiation, and 2) deciding whether the BA should proceed with or terminate the entire concurrent negotiation. Even though a BA may reach tentative agreements with PAs on some of the required resources while continuing to negotiate for better deals, it may not be prudent to postpone finalizing its contracts with PAs for some or all of its resources because other BAs may be competing for the same resources and PAs may break tentative contracts at any time. In the first stage of coordination, the coordinator module predicts the change in the expected outcome of each resource market using linear regression. If the combined expected outcome of all the resource markets in future negotiation rounds is favorable, the coordinator allows the concurrent negotiation to proceed to the next round. Otherwise, the
Each acceptor sends resource requiring providers.”

In [47], designed an agent-based multi-cloud scheduling model. Each agent has four modules: monitor, negotiator, scheduler, and executor. Triggered by task arrivals/completions or changes in resource availability, the monitor module gathers information about resource states and task status, and sends relevant information to the negotiator module. The negotiator module of an agent requires resources from other clouds broadcasts a task request to negotiator modules of all other agents. Each negotiator module of agents representing other cloud providers evaluates the request and decides whether to accept or reject the request based on its own cloud’s internal resource needs and the requirements of the task request. Each negotiator module that wishes to accept the request sends an estimated completion time for the task. The negotiator module with the shortest estimated completion time is selected to execute the task. On this account, “negotiator” modules can be viewed as analyzing resource needs and performing cloud resource selection rather than negotiation in the sense of making concessions and reaching mutually acceptable agreements. The scheduler module plans the mapping of tasks to resources. In [47], a scheduler module is attached to one or several resources belonging to a cloud provider. Since the scheduling platform in [47] strives to enable each cloud provider to maintain its own internal scheduling policies, a scheduler module can relocate tasks to other agents or reschedule tasks to other resource queues under its control. Scheduling policies are implemented as event-condition-action (ECA) rules and executed using a workflow orchestration engine called OSyRIS [48]. In [48], each rule has the construct “LHS->RHS|condition, salience” that specifies the conditions required by an event to trigger the execution of consequent actions. Whereas LHS represents the set of tasks that needs to be executed before the tasks in RHS, condition and salience specify the conditions and priorities of rules, respectively. Salience is represented by an integer, with larger values representing higher priorities. The scheduler module has a rule base which can be modified if there are changes to the scheduling policies. OSyRIS will execute all the rules with conditions that match the data in its working memory. Data in the working memory is updated upon the execution of rules, which may in turn trigger further executions of other rules. A chained execution of a series of ECA rules provides an ordering plan for assigning resources to tasks. For example, in the set of ECA rules in Fig. 1, “a” (respectively, “b”) binds the output of task A (respectively, B) to the input of task B (respectively, C) and “c” binds the output of task C to the inputs of both tasks D and E. However, E (having a higher salience value) will be executed before D. (Details and other examples of ECA rules in OSyRIS can be found in [49].) Tasks to be executed by a resource are placed in that resource queue by the scheduler module. Finally, the executor module extracts tasks from resource queues for execution.

![Fig. 1. A Set of ECA Rules in OSyRIS](image)

---

**2.5 Scheduling and Workflow**

This sub-section reviews two approaches for determining the sequence of service components.

1) *Cloud Scheduling using ECA Rules*: Implementing agents as a *monitor-analyze-plan-execute (MAPE)* control loop, Frincu et al. [47] designed an agent-based multi-cloud scheduling model. Each agent has four modules: monitor, negotiator, scheduler, and executor. Triggered by task arrivals/completions or changes in resource availability, the monitor module gathers information about resource states and task status, and sends relevant information to the negotiator module. The negotiator module of an agent requiring resources from other clouds broadcasts a task request to negotiator modules of all other agents. Each negotiator module of agents representing other cloud providers evaluates the request and decides whether to accept or reject the request based on its own cloud’s internal resource needs and the requirements of the task request. Each negotiator module that wishes to accept the request sends an estimated completion time for the task. The negotiator module with the shortest estimated completion time is selected to execute the task. On this account, “negotiator” modules can be viewed as analyzing resource needs and performing cloud resource selection rather than negotiation in the sense of making concessions and reaching mutually acceptable agreements. The scheduler module plans the mapping of tasks to resources. In [47], a scheduler module is attached to one or several resources belonging to a cloud provider. Since the scheduling platform in [47] strives to enable each cloud provider to maintain its own internal scheduling policies, a scheduler module can relocate tasks to other agents or reschedule tasks to other resource queues under its control. Scheduling policies are implemented as event-condition-action (ECA) rules and executed using a workflow orchestration engine called OSyRIS [48]. In [48], each rule has the construct “LHS->RHS|condition, salience” that specifies the conditions required by an event to trigger the execution of consequent actions. Whereas LHS represents the set of tasks that needs to be executed before the tasks in RHS, condition and salience specify the conditions and priorities of rules, respectively. Salience is represented by an integer, with larger values representing higher priorities. The scheduler module has a rule base which can be modified if there are changes to the scheduling policies. OSyRIS will execute all the rules with conditions that match the data in its working memory. Data in the working memory is updated upon the execution of rules, which may in turn trigger further executions of other rules. A chained execution of a series of ECA rules provides an ordering plan for assigning resources to tasks. For example, in the set of ECA rules in Fig. 1, “a” (respectively, “b”) binds the output of task A (respectively, B) to the input of task B (respectively, C) and “c” binds the output of task C to the inputs of both tasks D and E. However, E (having a higher salience value) will be executed before D. (Details and other examples of ECA rules in OSyRIS can be found in [49].) Tasks to be executed by a resource are placed in that resource queue by the scheduler module. Finally, the executor module extracts tasks from resource queues for execution.

\[
\begin{align*}
A[a=01] & \rightarrow B[b=01]. \\
B[b=01] & \rightarrow C[c=01]. \\
C[c=01] & \rightarrow D[d=01]. \\
D[d=01] & \rightarrow E[e=01].
\end{align*}
\]

---

2) *Workflow using Nested Petri-nets*: Supporting Intercloud workflow execution involves allocating and composing a collection of (heterogeneous) resources from different clouds and coordinating tasks execution using these resources, taking ordering constraints into consideration. Using colored Petri-nets (CPNs) [50,51] and nested Petri-nets (NPNs) [52] (see Appendix D in supplemental material) to synchronize and coordinate concurrent workflow, Gutierrez-Garcia and Sim [53] devised an agent-based approach for supporting workflow execution involving resources from multiple clouds. In a CPN, tokens have colors representing different data types and arcs can have logical expressions which prevent transitions from firing if they are evaluated as “false” (Fig. 2). In [53], whereas service composition is carried out using the *contract net protocol* (CNP), Intercloud workflow execution is modeled using two levels of NPN: 1) *system nets* for modeling agent behaviors and 2) *object nets* for modeling cloud workflows (CWs). An NPN may contain both colored and uncolored tokens and have both local transitions and synchronized transitions. Local transitions (e.g., “T” in Fig. 2) create, add, and remove tokens at the level where the transitions are located. Synchronized transitions (e.g., “ST” in Fig. 2 and “ST” in Fig. 3) extend local transitions by adding binding constraints (conditions for firing the synchronized transitions) between object nets (inner-level nets) and system nets (outer-level nets). A CW is a series of interrelated tasks to be performed by possibly different clouds.

In the example described in this sub-section, the behaviors of a BA are modeled as a system net (Fig. 2) for coordinating the execution of CW modeled as an object net. Figs. 3 and 4 show the synchronization module and the main structure of CW, respectively.

Fig. 2 depicts a firing sequence \( \sigma = \{T_5, T_3, ST_4\downarrow, T_4, ST_4\uparrow, ST_3\downarrow, T_1\} \) that represents the simplest coordination process among one CA, one BA, and \( n \) PAs. The start of the process is triggered by the event \( T_i \) in which BA receives a call for proposals from CA to execute a CW. This leads to the execution of the CNP with CA as the initiator and BA as a participant (\( P_i \)). There are two possible outcomes for the CNP execution: 1) BA accepts the contract from CA to execute CW (i.e., \( P_i \) has a token) or 2) BA rejects the contract (i.e., \( P_i \) has no token). As shown in Fig. 2, the token color (data type) in \( P_i \) is CW. Assuming that BA
accepts CA’s contract (otherwise, the process stops at P1 and whatever happens next is no longer the concern of BA). With a token in P1, T1 is enabled for firing. T1 represents the event of triggering the execution of CW. When T1 is fired, P1 has a token. Similar to P1, the token color in P3 is CW. This enables ST1↓ for firing. ST1↓ represents the event of BA sending a call for proposals to n PAs to fulfill a task of CW. This leads to the execution of the CNP with BA as the initiator and PAs as participants (P3). There are two possible outcomes for the CNP execution: 1) there is a PA that accepts the contract from BA to fulfill a task of CW (i.e., P3 has a token) or 2) all PAs reject the contract (i.e., P3 has no token). In case no PA accepts BA’s contract, BA will recursively restart the CNP execution. The token color in P3 is (Task, Req) which represents the task to be executed and the service required to execute the task, respectively. Assume that a PA accepts BA’s contract. With a token in P3, T3 is enabled for firing. T3 represents the event of PA sending the output from executing the task to BA. When T3 is fired, P3 has a token. The token color in P3 is (Task, Req, Output) which represents the task to be executed, the service required to execute the task, and the output from executing the task, respectively. The logical expressions in the output arcs of P3 will determine which transition ST3↓ or ST3↑ will be fired. In the event that PA fails to fulfill the task requirement (i.e., output.status=false), ST3↓ is triggered so that BA repeats the call for proposals to other PAs. ST3↓ will be fired if PA successfully fulfilled the task requirements (i.e., output.status=true). ST3↑ represents the event of BA updating the progress of CW (deleting tasks in the task list of CW that have been fulfilled). The firing of ST3↓ places a token in P3. If all task requirements in CW have been fulfilled, i.e., the output of CW is ready for delivery, then ST3↑ will be fired. This will place a token in P3. The token color in P3 is (CW, Output) which represents the workflow and the output from executing CW, respectively. Finally, T4 represents the event of BA sending the output from executing CW to CA.

In [53], each CW is managed by a BA. Whereas a BA selects and composes cloud services for executing all workflow tasks, a CW provides the ordering constraints for service composition. The synchronization module of CW (the object net) synchronizes the execution of CW with the behaviors of a BA (the system net). Synchronized transitions ST1↓, ST2↓, and ST3↓ of the system net (Fig. 2) have an inner-level (↓) binding constraint to enable synchronized firing with the object net. The synchronization module of the object net (Fig. 3) contains the corresponding synchronized transitions ST1↑, ST2↑, and ST3↑ that have an outer-level (↑) binding constraint to enable synchronized firing with the system net.

ST1↑ synchronizes the submission of workflow tasks with the transmission of a call for proposals from a BA to PAs (ST1↓). The token color of the input place P0 of ST1↑ is (Task, Req) which represents the task to be executed and the service required to execute the task, respectively. In Figs. 2 and 3, synchronized transitions ST↓ and ST↑ are fired in both the system net and the object net, respectively, when 1) the only input place P0 of ST↑ in the object net has a (Task, Req) token (i.e., there is a task in CW for execution), and 2) ST↓ of the system net has CW as its token in its only input place P0. The firing of ST↑ places one (Task, Req) token in Pc as a record for the execution-in-progress of a task in CW.

---

**Figure 2:** A Broker Agent’s Behaviors (a System Net) [53]

**Figure 3:** Synchronization Module of a Cloud Workflow [53]
In the event that PA fails to fulfill the task requirement (i.e., output.status=false), \(<ST\rangle\uparrow\) synchronizes the re-submission of a workflow task with the event that BA repeats the call for proposals to other PAs (\(<ST\rangle\downarrow\)). In Figs. 2 and 3, synchronized transitions \(<ST\rangle\downarrow\) and \(<ST\rangle\uparrow\) are fired in both the system net and the object net, respectively, when 1) the only input place \(P_c\) of \(<ST\rangle\uparrow\) in the object net has a (Task, Req) token, and 2) \(<ST\rangle\downarrow\) of the system net also has a (Task, Req, Output) token in its only input place \(P_i\) and if PA fails to fulfill the task requirement. The firing of \(<ST\rangle\uparrow\) places one (Task, Req) token in \(P_i\) (i.e., re-submitting a workflow task).

\(<ST\rangle\uparrow\) synchronizes the execution of a workflow task with the event of BA updating the progress of CW (\(<ST\rangle\downarrow\)). In Figs. 2 and 3, synchronized transitions \(<ST\rangle\downarrow\) and \(<ST\rangle\uparrow\) are fired in both the system net and the object net, respectively, when 1) the only input place \(P_c\) of \(<ST\rangle\uparrow\) in the object net has a (Task, Req) token, and 2) \(<ST\rangle\downarrow\) of the system net also has a (Task, Req, Output) token in its only input place \(P_i\) and if PA successfully fulfilled the task requirement (i.e. output.status=true). The firing of \(<ST\rangle\uparrow\) places a token in an output place of \(<ST\rangle\uparrow\) that corresponds to the task id of the task that has been executed. This is determined by the logical expressions in the output arcs of \(<ST\rangle\uparrow\).

In Fig. 4, whereas \(T_s\) simply starts the execution of CW, \(<ST\rangle\uparrow\) delivers the output of CW. Both \(<ST\rangle\downarrow\) of the system net and \(<ST\rangle\uparrow\) in the main structure of the object net are synchronized transitions. Whereas \(<ST\rangle\downarrow\) has an inner-level (\(\downarrow\)) binding constraint to enable synchronized firing with the object net, \(<ST\rangle\uparrow\) has an outer-level (\(\uparrow\)) binding constraint to enable synchronized firing with the system net. In Figs. 2 and 4, synchronized transitions \(<ST\rangle\downarrow\) and \(<ST\rangle\uparrow\) are fired in both the system net and the object net, respectively, when 1) all \(m\) places connected to the \(m\) input arcs of \(<ST\rangle\uparrow\) in the main structure of the object net have a token and 2) \(<ST\rangle\downarrow\) of the system net has CW as its token in its only input place \(P_i\).

![Fig. 4. Main Structure of a Cloud Workflow [53]](image)

### 3. COMPARISON AND CRITIQUE

This section compares the state-of-the-art agent-based Intercloud resource allocation models reviewed in section 2. Table 1 summarizes and classifies agent-based models for cloud resource discovery and matching, selection and composition, monitoring, and scheduling and workflow in terms of their 1) Intercloud models, 2) interaction protocols/mechanisms, and 3) resource allocation activities. Table 2 compares agent-based models for cloud resource negotiation in terms of 1) negotiation patterns, 2) concession-making strategies, 3) negotiation interactions, and 4) number of markets.

#### 3.1 Agent-based Intercloud Interaction Protocols

This sub-section compares the agent-based interaction protocols/mechanisms for cloud resource discovery, matching, selection, composition, monitoring, scheduling, and workflow in section 2 (see Table 1).

1) Brokering vs. matching: Interactions between agents in both [6] and [23] are mediated by intermediaries. Whereas BAs in [23] compare requests from CAS with advertisements from PAs before matching requests to advertisements, the message broker in [6] stores messages from publisher DAs, filters messages by selecting messages according to the message class subscriptions, then forwards the relevant messages to subscriber DAs. However, while agents in [23] were designed to support consumers in discovering cloud resources, agents in [6] were designed to support Intercloud resource discovery. In [6], MAs match FCs to HCds by aligning two sets of rules representing requirements of HCs and policies of FCs. In [23], BAs connect CAS to PAs through similarity reasoning and price-and-time-slot matching.

2) FSCNP vs. CNP: In CNP [27,28], an agent attempts to select the services of other agents by broadcasting its requests to all agents in the system. In FSCNP [11], an agent consults an SCT, then focuses its service selections by interacting only with those agents that provide the relevant resources or services. In terms of the number of messages exchanged among cloud agents, mathematical analyses in [11,30] show that the FSCNP is more efficient than the CNP. In FSCNP, it is assumed that the SCTs are given. However, to date, there is no work explicitly addressing the issues of constructing and maintaining accurate SCTs. Recording and updating information about resource states require continuous resource monitoring such as those in [31-33].

Agents in [26] and [29] adopting the CNP can only match a job to an individual resource. Designed for multiple concurrent subcontracting interactions, agents in [30] adopting the FSCNP can combine a set of resources from multiple providers and deliver the combined resources as a single unified service. Furthermore, a cloud agent using 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.

Additionally, resource allocation models in [26,29,30] only allow agents to have a one-shot selection of the resources, and there is no negotiation mechanism for optimizing resource utilization or price such as those in [35,36,42,44,45].

3) Active vs. passive monitoring: From the perspective of multiagent systems, active monitoring in [31,32] is preferred over passive monitoring [33] since ACAs are designed with a higher degree of autonomy, i.e., they can decide when to proactively send information about the states of their resources to ACMs. However, even though Endo et al. [32] proposed a framework for active resource monitoring, they did not devise a communication protocol for ACAs and ACMs to share information. On the other hand, agents in [33] adopt the CPRP to adaptively adjust...
time intervals for checking resource states based on different check responses. Whereas CPRP can reduce the number of messages exchanged by adaptively adjusting the time intervals for checking resource states based on different responses, there may be a tradeoff in the accuracy of the resource states due to abrupt and unforeseen failures.

4) Nested Petri Nets vs. Event-condition-action rules: Whereas agents in [53] manage Intercloud workflow execution using coordination by synchronization [24, p.409], agents in [47,48] manage Intercloud workflow execution using coordination by planning [24, p.410]. Coordination by planning consists of: 1) producing a set of plans by considering the set of actions to be carried out and 2) selecting actions from the plans for execution. Workflow management in [47,48] consists of 1) creating an ordering plan for assigning resources to tasks through a chained execution of ECA rules and 2) extracting tasks from resource queues for execution. Using ECA rules enables adaptation to changes in scheduling policies through modifications of ECA rules in the rule base. Using CPNs [50,51] and NPNs [52] to synchronize the task executions of Intercloud workflow (the object net) with actions of a society of agents (the system net) provides designers with two advantages: 1) systems visualization (having an overview of the parallel executions of concurrent and interrelated processes) and 2) formal semantics (having a widely accepted formal foundation for modeling concurrency and synchronization [50]).

<table>
<thead>
<tr>
<th>Summary of Agent-based Cloud Computing Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercloud model</td>
</tr>
<tr>
<td>Celesti et al. [6]</td>
</tr>
<tr>
<td>Ejarque et al. [26]</td>
</tr>
<tr>
<td>Venticinque et al. [29]</td>
</tr>
<tr>
<td>Wei &amp; Blake [33]</td>
</tr>
<tr>
<td>Endo et al. [31][32]</td>
</tr>
<tr>
<td>Gutierrez &amp; Sim [53]</td>
</tr>
</tbody>
</table>

3.2 Agent-based Cloud Negotiation Models

This sub-section compares the negotiation patterns, concession-making strategies, and interactions and market structures of agent-based cloud negotiation models in section 2 (see Table 2).

1) Negotiation patterns: Adopting the one-shot negotiation pattern, agents in [35] generally do not have a chance to modify their proposals in the hope of reaching an agreement. Adopting the alternating negotiation pattern, agents in [36,42,44,45] are allowed to make a series of proposals. As such, they can revise their proposals by making concessions in the hope of improving their chance of reaching an agreement. Whereas agents in both [36] and [42] adopt the alternating offers protocol, agents in [44] adopt a two-phase protocol, first exploring proposals that are of mutual interest in the “warm-up” phase, then finding more appropriate proposals in the “countdown” phase. Designed for concurrent negotiation of cloud resource co-allocation, agents in [45] adopt a complex but flexible negotiation protocol with an alternating revocable negotiation pattern in which each agent can be freed from a contract by paying a penalty fee. In cloud resource co-allocation, there are two advantages for allowing agents to be freed from contracts. First, if a BA fails to acquire all its required resources before its deadline, it can release those resources acquired so that PASs can (re-)assign them to other BAs. Second, a BA that has already reached an intermediate contract for a resource can continue to search for better deals before the entire concurrent negotiation terminates.

2) Concession-making strategies: Although agents in [36], [42], and [44] were designed for many-to-many negotiations, they adopt the time-dependent (T-D) strategy with fixed concession rates and do not consider market-oriented issues such as market rivalry (competition) and outside options (opportunity) [38-40] (see Appendix C). Adopting the BPE strategy, agents in [45] were specially designed to respond to different market conditions by making adjustable amounts of concession. Empirical studies were carried out in [45] comparing the performance of the BPE and T-D strategies under different market conditions. Empirical results show that agents adopting the BPE strategy achieved significantly higher utilities (i.e., better negotiation outcomes) than agents adopting the T-D strategy [45]. Agents adopting the BPE strategy obtained higher utilities because they are more prudent in making concessions, i.e., they do not make excessive (respectively, inadequate) amounts of concession in favorable (respectively, unfavorable) markets. For example, when there are more CAs competing for services, a BA is in a favorable market. Since the BA has a relative strong BP, it will make smaller amounts of concessions.

<table>
<thead>
<tr>
<th>A Comparison of Cloud Negotiation Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiation pattern</td>
</tr>
<tr>
<td>Venticinque et al. [35]</td>
</tr>
<tr>
<td>Son &amp; Sim [36]</td>
</tr>
<tr>
<td>Dastjerdi &amp; Buyya [42]</td>
</tr>
<tr>
<td>Siebenhaar et al. [44]</td>
</tr>
<tr>
<td>Sim [45]</td>
</tr>
</tbody>
</table>

3) Interactions and Cloud Market Structures: Interactions
among agents can be classified in terms of the number of participants in a negotiation [8, p.316]. In [35], (respectively, [36] and [42]), one-to-many (respectively, many-to-many) negotiations are carried out between two types of agents in one cloud market. In both [44] and [45], the negotiation mechanisms have much more complex structures where parallel many-to-many negotiations are carried out in multiple cloud markets among three types of agents. Furthermore, [45] modeled negotiations in multiple interrelated markets in which an agent’s negotiation outcomes in one market can potentially influence its negotiation outcomes in another market. For example, a BA receiving proposals with higher resource prices from PAs in multiple cloud resource markets will set higher prices in their proposals when negotiating with CAs in the cloud service market.

4. AGENT-BASED VS. NON-AGENT-BASED CLOUD BOTS EXECUTION TOOLS

Bag-of-tasks (BoT) applications are composed of multiple independent tasks with no execution ordering constraints, and can be highly parallelized for execution in multiple clouds. Section 4.1 reviews “CloudAgent” [54], an agent-based cloud BoT execution tool, and compares it with other non-agent-based cloud BoT execution tools, including Aneka [55,56] and SwinDeW-C [57]. Whereas section 4.2 describes the advantages of the agent-based approach over the centralized approach adopted by Aneka, section 4.3 discusses the advantages of the agent-based approach over the peer-to-peer approach adopted by SwinDeW-C.

4.1 CloudAgent

Designed to concurrently execute BoTs in a multi-cloud, CloudAgent’s architecture consists of consumer agents (CAs), broker agents (BAs), service provider agents (SPAs), resource agents (RAs), and a cloud directory [54]. Cloud BoT execution in CloudAgent is supported by a four-stage agent-based protocol consisting of: 1) establishment of cloud storage and agents’ communication channels, 2) resource provisioning, 3) BoT execution, and 4) resource de-allocation.

Cloud storage and communication channels: To store a BoT’s inputs and outputs, a CA establishes cloud storage by creating an S3 bucket [58] (i.e., a file container). Each CA uploads all the task input files to the S3 bucket. When it receives the public IP address of a resource from a BA, the CA adds access policies based on the resource’s IP address to the S3 bucket so that only the RA representing the authorized resource can download and upload files from and to the S3 bucket. The interactions between CAs and RAs are supported by Amazon Simple Queue Service (SQS) [59]. CAs establish SQS communication to bolster agent-based cloud BoT execution by creating two SQS queues: one for CAs to send messages to RAs, and the other for CAs to receive messages from RAs.

Resource provisioning: Resource provisioning consists of four phases as follows.

1) By adopting the CNP, each CA (playing the role of a manager) contracts BAs (playing the role of contractors) to manage the allocation of a set of required resources.

2) By adopting a parallel CNP, each BA (playing the role of a manager) in turn attempts to subcontract the CA’s resource requests to a group of SPAs (playing the role of contractors). For each resource requirement, a BA initiates a subcontracting process using the CNP to select SPAs by matching the CA’s requirements with capabilities of SPAs. With n resource requirements, a BA concurrently initiates n parallel subcontracting processes by adopting a parallel CNP.

3) SPAs selected by a BA allocate the corresponding resources and send the public IP addresses and passwords of the resources to the BA.

4) On receiving the IP addresses and passwords, the BA forwards them to the CA.

BoT execution: BoT execution consists of five phases as follows.

1) Assigning tasks to resources using the first-come-first-served scheduling heuristics (other scheduling heuristics [60] can also be used), a CA selects an unexecuted task of a BoT for execution.

2) To assign a task to a resource, the CA sends a task execution request to an RA with an unoccupied resource. A task execution request contains a task id, the location of its input files, and the expected location of its output files.

3) The RA downloads the input files from an S3 bucket, executes the task, then uploads its output files to the bucket.

4) The RA sends an “inform-done” message containing the task id of the task that has been executed to the CA.

5) The CA tags the task as executed, and continues to select another unexecuted task for execution.

Resource de-allocation: When a CA detects that it has no remaining unexecuted tasks to assign to a resource, a de-allocation event is triggered. A CA sends a de-allocation request containing the resource’s public IP address to the BA, which forwards the de-allocation request to the corresponding SPA. The SPA de-allocates the resource and sends a “confirm” message to the BA, which in turn forwards it to the CA.

4.2 Aneka vs. CloudAgent

Aneka [55,56] is a framework for developing, deploying, and managing cloud applications. It consists of a middleware deployed on top of heterogeneous cloud computing resources [55]. The building block of the middleware is Aneka container, which provides three classes of services: 1) execution services for scheduling and executing applications, 2) foundation services for allocating resources, managing the collection of available nodes, and maintaining the recency of the services registry, and 3) fabric services for providing access to resources managed by Aneka cloud (a collection of internetworked nodes). A typical deployment scenario of Aneka consists of 1) one master node that has capabilities for resource management, scheduling of applications, and access control to resources in the clouds and 2) one or more worker nodes that process tasks that compose the application [55,56].
Cloud BoT execution systems can generally be classified into centralized and distributed systems [54]. In a centralized cloud BoT execution system, a main control entity manages every subordinate entity and every process (e.g., resource provisioning) involved in a BoT execution. In distributed cloud BoT execution systems, entities with different capabilities interact among themselves for coordinating the executions of BoTs without any central authority. Whereas Aneka is a centralized BoT execution system consisting of multiple worker nodes that are fully controlled by a central master node that dynamically provisions resources from multiple providers based on administrator-defined allocation policies, CloudAgent is a distributed BoT execution system. Since the components in a centralized federated cloud are controlled by a central authority, while providing services to others, nodes in Aneka operate with very limited or no independence. In contrast, being autonomous entities, agents in CloudAgent operate with a high degree of flexibility, enabling each cloud to exert a greater control over its own resources and to have a higher degree of freedom in implementing its own resource management policies.

4.3 SwinDeW-C vs. CloudAgent

SwinDeW-C [57] is a peer-to-peer BoT execution system consisting of coordinator and worker peers that coordinate among themselves to execute BoTs in multi-cloud environments. Organized into a peer-to-peer federated cloud architecture consisting of peer groups, each peer group is managed by a coordinator peer. Whereas a worker peer is an ordinary cloud service node with certain service capabilities, a coordinator peer is a “super” node with both service capabilities and peer management capabilities (e.g., managing the joining, leaving, and searching of peers in the peer group through peer-to-peer communication). In SwinDeW-C, BoT specifications are submitted to coordinator peers that in turn distribute the tasks in the BoT to suitable peers through peer-to-peer communication. Unlike resource provisioning in Aneka which is controlled by a central entity, resource provisioning in SwinDeW-C is coordinated by a set of coordinator peers that distribute tasks to other peers based on their capabilities and loads.

However, even though both CloudAgent and SwinDeW-C are distributed BoT execution systems, they differ in the following ways. Organized as a system hierarchy, coordinator peers in SwinDeW-C have full control over worker peers. Agents in CloudAgent are autonomous in the sense that they are capable of making independent decisions. In SwinDeW-C, all the computing elements are implicitly assumed to share a common goal (of making the overall system function correctly), and tasks are redistributed and assigned to suitable peers. In a multiagent system such as CloudAgent, agents are self-interested, i.e., primarily concerned with their own welfare, and the decision to accept a task request lies solely with each agent. For example, in CloudAgent, an SPA may not accept a job request if it needs the resources to execute its own tasks or if the financial gain from performing the task is not sufficiently attractive. In this sense, CloudAgent was designed to provide the flexibility for each cloud to exert a greater control over its own resources and to have a higher degree of freedom in implementing its own resource management policies.

Furthermore, in a multiagent system such as CloudAgent, the encounters among agents are economic encounters. For its resource provisioning, CloudAgent adopts the CNP, which is a market-like protocol (a protocol for the drawing up of contracts in public markets) [24, p.359]. An example scenario was given in [54]. The CNP starts with a CA sending a CFP to a group of BASs to handle the allocation of a set of heterogeneous resources containing pre-installed data mining applications. Three BAS: BA1, BA2, and BA3 compete to provide the service by submitting their bids $[BA_1, \$9.75]$, $[BA_2, \$9.99]$, and $[BA_3, \$8.10]$. The CA selects the proposal with the lowest price, and awards the contract to BA1. In summary, using a market mechanism to set resource prices helps to regulate the supply of and demand for cloud resources and offers an efficient means for managing resources. For example, adopting the CNP, SPAs in CloudAgent can choose to bid for jobs that will maximize their profits while CAs and BASs can select services with the minimum costs.

5. CONCLUSION AND FUTURE DIRECTIONS

In this survey paper, an exposition of agent-based problem-solving approaches for intelligent Intercloud resource allocation is provided. The contributions of this paper are manifold.

1) Section 1 defines agent-based cloud computing and cloud intelligence. It describes the motivation, advantages, and significance of adopting an agent paradigm for intelligent Intercloud resource allocation.

2) Section 2 provides a comprehensive overview of the state-of-the-art research on adopting an agent-based paradigm for Intercloud resource allocation by reviewing representative agent-based Intercloud resource allocation models.

3) Section 3 provides a comparison and critique of the state-of-the-art agent-based Intercloud resource allocation models. Summarizing and comparing the features of existing agent-based Intercloud resource allocation models provide designers with pointers to and guidelines on some of the essential design considerations for developing new agent-based techniques for Intercloud resource allocation.

4) Section 4 provides a comparison between agent-based and non-agent-based approaches for cloud BoT execution. In particular, section 4 identifies and discusses the advantages of adopting an agent paradigm for Intercloud resource allocation.

5) Section 5 provides pointers to future directions. Whereas the IEEE Cloud Computing Initiative aims to create the IEEE Intercloud testbed to tie all clouds together [61] and the IEEE P2302 standard [62] for specifying Intercloud interoperability, it is anticipated that ABCC will play a significant role in another important aspect—shaping the “intelligent Intercloud” vision [14]. An intelligent Intercloud is an interconnected “cloud of clouds”
populated by a society of agents that automate interactions among stakeholders (consumers, brokers, and providers) in a wide range of resource allocation activities.

Some future directions of ABCC may include: 1) performing complexity and overhead analyses of agent-based cloud computing models reviewed in Section 2, 2) devising a communication protocol for cloud agents in active resource monitoring, 3) empirically comparing active and passive resource monitoring, 4) devising approaches for constructing and maintaining SCTs, 5) building learning cloud agents [63], and 6) designing a multi-layer Intercloud workflow Petri net [64], which are described as follows.

Except [11] and [30], all other agent-based cloud computing models reviewed in Section 2 did not provide any complexity and overhead analyses of their agent-based approaches (see Section 3.1).

As noted in Section 3.1, even though active monitoring is preferred over passive monitoring, a communication protocol for cloud agents in active resource monitoring is yet to be devised. Additionally, empirical studies comparing the performance of both active and passive resource monitoring are yet to be carried out.

To date, approaches for constructing and maintaining SCTs to bolster cloud service composition are yet to be devised (see Section 3.1).

Since cloud agents operate in a dynamically changing environment (e.g., changing user demands), it seems prudent to design learning cloud agents [63] that gather information (e.g., for predicting supply-and-demand patterns for resources) to assist cloud providers in making better resource management decisions.

Another new direction by Bendoukha et al. [64] may inspire researchers to design Petri nets for modeling Intercloud workflow at three layers: 1) user application layer, 2) middle layer, and 3) resource layer.

In the big data era, situations where applications require huge amounts of computing resources that can only be supplied by a federation of clouds will become increasingly common. The author hopes that this survey will raise the awareness of the advantages of ABCC among researchers and inspire them to take up future challenges of developing new agent-based techniques for bolstering intelligent Intercloud resource allocation in this big data era.

ACKNOWLEDGMENT

The author would like to thank the Editor-in-Chief, the Associate Editor, and the anonymous referees for their comments and suggestions.

REFERENCES


IEEE TRANSACTIONS ON CLOUD COMPUTING


Kwang Mong Sim (FIET’16, SM’07) was a Chair (Professor) in Computer Science at the University of Kent, UK. Professor Sim is a Fellow of the IET. He is the Associate Editor-in-Chief of Springer’s Applied Intelligence Journal and an Associate Editor of both the IEEE Transactions on Cloud Computing and the IEEE Transactions on Cybernetics. He was an Associate Editor of the IEEE Transactions on Systems, Man and Cybernetics-Part C for ten years. Prof. Sim has published close to 160 technical papers, including ACM Computing Review Best of 2013 paper and IEEE Transactions spotlight paper, frequently-cited IEEE Transactions survey papers, and best papers in international conferences. He has delivered many keynote lectures on intelligent Cloud computing, agent-based Cloud computing and automated negotiation in many international conferences. Prof. Sim received his PhD and MSc from the University of Calgary, AB, Canada, and graduated Summa Cum Laude with a BSc(Hon) from the University of Ottawa, ON, Canada.