Minimizing Message Exchanges in Agent Based Cloud Service Composition

Suleiman Onimisi Aliyu and Kwang Mong Sim, IEEE Senior Member

***Abstract*—Previous work demonstrated that by adopting a semi-recursive contract net protocol (SR-CNP) equipped with service capability tables (SCTs) for dynamically selecting recorded cloud agents, their services and states, Cloud agents can effectively integrate disparate Cloud resources into a unified Cloud service. However, the choice of SCT may result in large overheads with Cloud agents exchanging a considerably large number of messages to achieve high success rates in service composition. In this paper, a comprehensive set of mathematical analyses of message exchanges by Cloud agents (Broker agents, Consumer agents and Service provider agents) in cloud service composition are presented. Experiments were performed where cloud agents adopt Particle Swarm Optimization for evolving the best service composition outcomes with the aim of minimizing the average number of messages propagated while successfully composing cloud services using SR-CNP and SCTs. Empirical results obtained from an agent-based testbed reveal that agents successfully minimized the number of messages exchanged during cloud service composition.**

***Index Terms*—agent based cloud service composition, multiagent systems, Cloud computing, contract net protocol, Cloud service composition.**

I. INTRODUCTION

CLOUD computing pools a set of web-accessible resources, provisioned under service level agreements and established via negotiation. Furthermore, it should be dynamically composed and virtualized according to consumers' need on an on-demand basis [1].

Over the past few years, Cloud service providers (e.g. Microsoft [2], Amazon [3], Google [4], Go Grid [5], IBM [6] etc.) have continued to evolve and the Cloud services churned out by these providers have continued to increase proportionately. On the other hand, the complexity of Consumers (e.g. App developers) requirements have also changed over time. In order to satisfy these complex consumer requests as they emerge, there is need for a dynamic and automated service composition that can support everything as-a-service model [7]. Hence, cloud service composition in single or multi-cloud environments must support the coordination of independent and self-interested parties, efficient re-configuration of existent and permanent service compositions since consumer requirements can change, dynamic and automated composition of distributed and parallel services, being able to deal with incomplete information about cloud participants as in a distributed system and service selection based on dynamic market-fees of cloud services[1], [8], [9].

An agent is a computational entity that acts on behalf of another entity (or entities) to perform a task or achieve a given goal. Agent systems are self-contained software programs embodying domain knowledge and having ability to behave with a specific degree of independence to carry out actions needed to achieve specified goals. Agent-based cloud computing is concerned with the design and development of software agents for bolstering Cloud service discovery, service negotiation and service composition [7].

There is a body of works in intelligent Cloud computing that strives to make Clouds more intelligent by adopting intelligent agents to automate the interactions among Clouds and between consumers and Cloud [10]. Agents within an intelligent InterCloud can automatically establish Cloud service contracts, integrate Cloud resources, coordinate concurrent Cloud workflow, and schedule parallel execution of tasks in multiple clouds. This work focuses on designing and building agents that can effectively and efficiently integrate computing resources from multiple Clouds.

In Agent-based cloud service composition, the real challenge is to assemble a single unified service dynamically from multiple cloud service providers [7]. Hence, it is imperative that Cloud service providers cooperate and harness each other's service capabilities to meet the demands of varying consumer requests.

Novel experiments in [7] for agent-based cloud service composition adopted the focused selection contract net protocol (FSCNP) as the agent interaction protocol and Service capability tables (SCTs), to record the list of Cloud agents, their services and current status. FSCNP is an extension of the contract network protocol (CNP), a distributed problem solving technique used for establishing service contracts among consumers and contractors.

Although Service capability tables (SCTs) are quite akin to the idea of acquaintance networks (ANs), SCTs record not only the service capabilities, but also, the state of cloud agents; and since an agent's state can change, SCTs are updated more often than ANs. In ANs, the capability tables may be updated only when new agents are joining the environment or existing agents depart the environment.

Results from experiments in [7] show agents can self-organize by using their own knowledge of agents listed in Service capability tables to successfully compose cloud services. However, the results also show that the success rates of the Cloud service composition depends on the level of knowledge that an agent about other agents' service

Manuscript received January 12, 2014.

S. O. Aliyu on study fellowship from Abubakar Tafawa Balewa University, Bauchi-Nigeria is currently a doctoral student in The School of Computing, University of Kent, Chatham Maritime, UK (e-mail: soaliyu@gmail.com).

K. M. Sim is with The School of Computing, University of Kent, Chatham Maritime, Kent, UK (e-mail: prof_sim_2002@yahoo.com).

.

capabilities. The tradeoff is that by having a higher level of knowledge of other agents' capabilities, more messages are exchanged causing more overheads.

The focus of this paper is on remodeling cloud agents to minimize message exchanges among Cloud agents. Herein, a Particle Swarm Optimization technique (PSO) has been applied to the problem formulation in order to minimize the average number of messages propagated whilst successfully composing cloud services using SR-CNP and SCTs.

For the purpose of concept verification, an agent based testbed has been utilized to generate empirical results. The results so obtained, showed that the agents successfully minimized the number of messages exchanged during cloud service composition.

The rest of the paper is organized as follows. Section II highlights contract net protocol for cloud service composition whilst Section III is entirely devoted to problem formulation and underlying mathematical analyses. For the sake of completeness, Section IV presents mathematical characterization of message utility for cloud composition agents. Brief description of the testbed is set forth in Section V, followed by simulation results in Section VI. Conclusion and future work can be found in Section VII.

II. CONTRACT NET PROTOCOL FOR CLOUD SERVICE COMPOSITION

An agent interaction protocol is a communication pattern among agents with potentially different roles to attain a specific design objective [11]. Basically, Agent interaction protocols govern the exchange of a series of messages among agents [12]. In the contract net protocol (CNP), an agent assumes one of two roles, a consumer (manager or Client) or a contractor (Server). Hence, using CNP, an agent requiring cloud services or resources of other agents would play the role of a manager and send call-for-proposal messages to all other agents. Agents providing cloud services or resources play the role of a contractor. Contract agents listen to call-for-proposals, evaluate a list of call-for-proposals as well as submit bids for contracts. Manager agents evaluate bids from contractor agents then select and award the contract to the most appropriate contractor agent based on its service capabilities [13].

The work reported in [7] devised FSCNP which differs from the classical CNP in that agents can assume multiple roles (Contractors and managers), agents can integrate results from multiple concurrent subcontracting interactions as well as being capable of recording the states of other agents. Theoretically, FSCNP is more efficient than CNP in terms of messages exchanged and from the empirical results in [7], agents following FSCNP that used strongly connected SCTs exchanged the most messages during service composition. This was because cloud agents following FSCNP select only relevant (and feasible) service providers (contractors) from service capability tables in an attempt to successfully compose cloud services. Hence, knowing more feasible agents would translate into sending more messages (on the average) using this protocol.

Gutierrez-Garcia and Sim in [1] proposed the Semi-recursive contract net protocol (SR-CNP) incorporating the feature of service capability tables (SCTs) to focus on selecting only feasible agents capable of accomplishing a task. Again, experimental results using SR-CNP show agents exhibiting a good sense of self- organization and cooperation in composing cloud services.

The Agent-based testbed for the experiments conducted in [1], [7] comprised web services (WSs), resource agents (RAs), service provider agents (SPAs), consumer agents (CAs), Broker Agents (BAs). By definition, a web service is a remotely accessible (preferably over the internet) software application or cloud resource. A web service can be viewed as the building block (atomic requirement) of a cloud service. A resource agent (RA) manages and controls access to a web service, i.e. a resource agent acts as a wrapper to a cloud service. Resource agents accept requests from its supervisory SPA or other sibling RAs to satisfy a service requirement and transmits service outputs to the requesting agent. RAs maintain a SCT of sibling RAs under the supervision of an SPA. The RA's SCT maintains the capabilities of sibling RAs, their location as well as their states. Service provider agents manage a cloud service provider's resources by managing and coordinating RAs under their administration.

To carry out its functions, an SPA is equipped with two SCTs- one for RAs under its administration and the other for SPAs. The latter is used only when RAs under its supervision cannot handle the requirements and may need the assistance of other RAs under the administration of another SPA. Broker agents' acts as mediators between CAs and SPAs, BAs provide a single virtualized service for CAs by composing a set of atomic requirements of the service from multiple SPAs. BAs maintain two SCTs, one for SPAs and the second for other BAs (To delegate requirement). Finally, Consumer agents submit consumer requests for a cloud service composition to BAs recorded in their SCT which in turn contact SPAs to compose the set of requirements contained in the service composition request.

III. MATHEMATICAL ANALYSES OF MESSAGE EXCHANGES IN AGENT BASED CLOUD SERVICE COMPOSITION

In this section, we present the mathematical analyses of message exchanges in agent-based cloud service composition for broker agent, consumer agent and service provider agent. We also made effort to establish the order by which messages exchanged by BAs, CAs and SPAs in service composition can be minimized (worst case analysis). The mathematical analyses of the aforementioned cloud agents are presented in the following subsections.

A. Broker agent (BA)

The number of messages exchanged is a critical performance measure in the complexity of cloud agents' behavior. The number of messages sent by the Broker Agent is tightly coupled to the level of connectivity of the agent's service capability table which, in turn should be determined by the perception of the agent in the environment. When a consumer agent submits a p-requirement request to a Broker agent, the broker agent in turn creates p– instances of the SR-CNPInitiatorBA behavior. For each instance, the BA with q feasible Service provider agents (SPA's) will act commensurately and send only s (s is the lower integer bound of $\alpha_{BA}q$, where $0 \leq \alpha_{BA} \leq 1$) call-for-proposals messages based on the perception of its environment. The BA then sends s messages (1 accept-proposal message and s-

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

1 reject proposal messages). If the contracted SPAs fail, the failure propagates to the BA. Then the BA then sends s-1 call-for-proposals messages to the remaining feasible SPAs and s-1 responses and so on. Therefore, in the worst case the BA sends:

$$p(2s + 2(s-1) + 2(s-2) + 2(s-3) \ldots + 2(2) + 2(1)) =$$
$$2p(s + (s-1) + (s-2) + (s-3) \ldots + (2) + (1)) =$$
$$2p(s(s+1)/2) = ps(s+1) \quad messages. \qquad \textbf{(1)}$$

Comparing this with a case where the BA with q feasible SPA's sends q call-for-proposals messages as so on, we propose the ratio τ measuring the degree of reduction in the messages exchanged,

$$\tau = pq(q+1)/ps(s+1) \qquad \textbf{(2)}$$

Since, $s \cong \alpha_{BA} q$ then we can rewrite (2) as,

$$\tau = pq(q+1)/p\alpha_{BA} q(\alpha_{BA} q + 1) \qquad \textbf{(3)}$$

Reducing (3) and rearranging terms, we have,

$$\tau = \frac{(q+1)}{\alpha_{BA}(\alpha_{BA} q+1)} = \frac{(q+1)}{(\alpha_{BA} q+1)} \cdot \frac{1}{\alpha_{BA}} \qquad \textbf{(4)}$$

Taking the limit of (4) as q tends to ∞ and α_{BA} tends to 0, it implies therefore,

$$\lim_{(\alpha_{BA}, q)\to(0,\infty)} \frac{(q+1)}{(\alpha_{BA} q+1)} \cdot \frac{1}{\alpha_{BA}} =$$

$$\lim_{(\alpha_{BA}, q)\to(0,\infty)} \frac{(1+1/q)}{(\alpha_{BA}+1/q)} \cdot \lim_{(\alpha_{BA}, q)\to(0,\infty)} \frac{1}{\alpha_{BA}} \to \infty^2.$$

Hence, the messages exchanged by BAs in [1], [7] in the worst case can be reduced by $O(n^2)$.

Similarly, the limit of (4) as q tends to ∞ and α_{BA} tends to 1 is,

$$\lim_{(\alpha_{BA}, q)\to(1,\infty)} \frac{(q+1)}{(\alpha_{BA} q+1)} \cdot \frac{1}{\alpha_{BA}} =$$

$$\lim_{(\alpha_{BA}, q)\to(1,\infty)} \frac{(1+1/q)}{(\alpha_{BA}+1/q)} \cdot \lim_{(\alpha_{BA}, q)\to(1,\infty)} \frac{1}{\alpha_{BA}} \to 1.$$

Consequently, the number of messages exchanged in the worst case for BAs in [1], [7] and the remodeled BAs are the same as α_{BA} tends to 1.

For example, if $\alpha_{BA} = 0.09$ *and* $q = 100$, a BA would send $0.09(100)(0.09(100)+1) = 9(10) = 90$ *messages* in the worst case by contracting only 9 feasible SPAs to satisfy a cloud service requirement as opposed to contracting all 100 feasible SPAs ($\alpha_{BA} = 1$), where the BA sends $1(100)(1(100)+1) = 100(101) = 10{,}100$ *messages* in the worst case in quests to secure a cloud service requirement as seen in Fig. 1.

In the event of failure from contracting SPAs, the BA, using its SR-CNPInitiatorBA behavior, can also subcontract fellow BAs in a manner similar to how they contract SPAs. We omitted the mathematical analysis of messages exchanged in the worst case because they are identical.

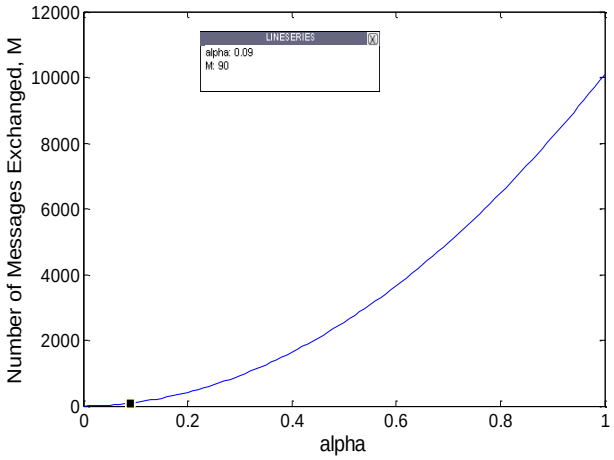

Fig. 1. Worst case analysis of messages exchanged by a BA contracting 100 SPAs in its SCT.

B. *Consumer Agent (CA)*

The Consumer agent (CA) in [1],[7] make requests for service composition by sending messages to Broker Agents recorded in their Service Capability Tables (SCTs). Therefore, if r BAs are recorded in their SCTs then the CA sends $2r$ messages in quest for service composition. The re-modelled CA will optimize the number of messages sent for service composition by sending a lower integer bound of α_{CA} r (α_{CA} is a value in [0, 1]) messages based on its perception of the service composition environment. Comparing the worst case scenarios of both implementations we have an expression ρ that measures the degree of reduction in messages exchanged:

$$\rho = \frac{2r}{2\alpha_{CA} r} \qquad \textbf{(5)}$$

Which simply becomes,

$$\rho = \frac{1}{\alpha_{CA}} \qquad \textbf{(6)}$$

Taking the limit of (6) as α_{CA} tends to 0, $\lim_{\alpha_{CA}\to 0} \frac{1}{\alpha_{CA}} \to \infty.$

Therefore, the messages exchanged by CAs in [1], [7] can be reduced by $O(n)$.

Evaluating the limit of (6) as α_{CA} tends to 1, $\lim_{\alpha_{CA}\to 1} \frac{1}{\alpha_{CA}} \to 1.$

The messages exchanged would be the same as CAs in [1], [7].

A Consumer Agent with $\alpha_{CA} = 0.2$ *and* $r = 100$ would engage 20 Broker Agents and in the worst case send 40 messages delegating a Broker Agent to satisfy a consumer requirement. Whereas, A Consumer Agent at position $\alpha_{CA} = 1.0$, would send at worst 200 messages involving all 100 feasible BAs to fulfill a cloud service request (see Fig. 2.).

C. *Service Provider Agent (SPA)*

A Service Provider Agent (SPA) sends a request message to a feasible Resource Agent (RA) to resolve a service requirement. If the RA fails, it then sub delegates another feasible RA to resolve the Cloud service requirement, if it

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

exists. Otherwise, it requests its supervisory SPA to delegate the request to feasible SPAs in its Service capability table (SCT) who can resolve such requirements using the SR-CNPInitiatorSPA behavior.

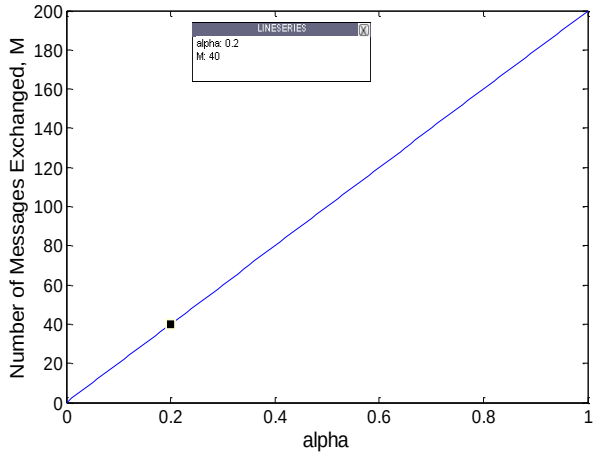

Fig. 2. Worst case analysis of messages exchanged by a CA for 100 BAs in its SCT.

The complexity of the behavior of the SPA in [1], [7] in the worst case scenario is $r(i + q(q + 1))$. Where r stands for the number of requirements handled by the SPA, i is the number of feasible RAs and q is the number of feasible SPAs recorded in its SCT. The modified SPA would instead of sending messages to q feasible SPA send a lower integer bound of $\alpha_{SPA}q$ (α_{SPA} is a value between 0 and 1) messages based on its perception of the capabilities of other SPAs in the environment using the SR-CNPInitiatorSPA behavior.

Comparing the worst case scenario of both implementations, we introduce a relation σ measuring the extent of reducing the number of messages exchanged:

$$\sigma = \frac{r(i+q(q+1))}{r(i+\alpha_{SPA}q(\alpha_{SPA}q+1))} \quad \textbf{(7)}$$

Simplifying (7) we obtain,

$$\sigma = \frac{(q^2+q+i)}{(\alpha_{SPA}{}^2q^2+\alpha_{SPA}q+i)} \quad \textbf{(8)}$$

Taking the limit of (8) as α_{SPA} tends to 0, that is, $\lim_{\alpha_{SPA}\to 0} \frac{(q^2+q+i)}{(\alpha_{SPA}{}^2q^2+\alpha_{SPA}q+i)}$ we get,

$$\sigma = \frac{(q^2+q+i)}{i} = 1 + \frac{q^2+q}{i} \quad \textbf{(9)}$$

From (9) as α_{SPA} tends to 0, q and i also tend to ∞, we obtain the limit, $\lim_{(q,i)\to(\infty,\infty)} 1 + (q^2/i) + (q/i) \to \infty$.

Hence, the number of messages propagated by SPAs in [1], [7] in the worst case can be minimized by $O(n)$.

Taking the limit of (8) as α_{SPA} tends to 1, q and i also tend to ∞, it implies therefore, $\lim_{(\alpha_{SPA},q,i)\to(1,\infty,\infty)} \frac{(q^2+q+i)}{(\alpha_{SPA}{}^2q^2+\alpha_{SPA}q+i)} \to 1.$

Thus, the number of messages exchanged are the same as SPAs subcontracting SPAs in [1], [7].

Figure. 3 illustrates graphically the worst case analysis of an SPA with 3,000 feasible RAs and 100 feasible SPAs in its SCT. An SPA at $\alpha_{SPA} = 0.09$, after all contracted RAs fail, would subcontract only 9 feasible SPAs to resolve a service requirement and send $3000 + 0.09(100)(0.09(100) + 1) = 3{,}090\ messages$. Conversely, an SPA subcontracting all 100 feasible SPAs to satisfy a cloud service requirement will send $3000 + 1(100)(1(100) + 1) = 13{,}100\ messages$.

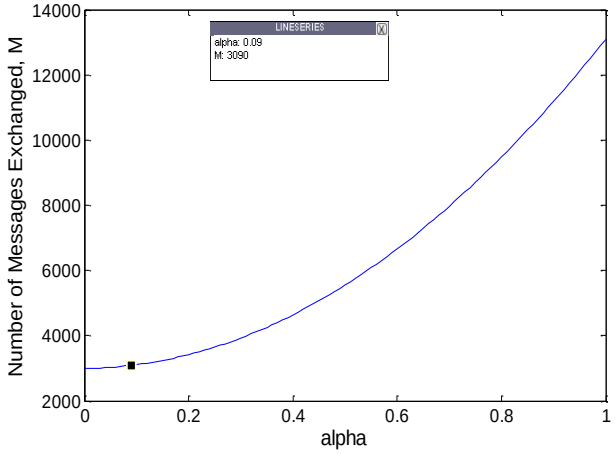

Fig. 3. Worst case analysis of messages exchanged by an SPA with 3,000 RAs and 100 SPAs in their SCT.

IV. MESSAGE UTILITY FOR CLOUD COMPOSITION AGENTS

We introduce the Message utility for cloud composition agents as a measure of how efficient (i.e. message-wise) agents are while successfully composing a cloud service or requirements of a cloud service. The message utility of cloud composition agents using service capability tables consists of a base utility (φ) and a message position utility ($1 - \beta$). The base utility is the minimum reward a cloud composition agent gets for successfully composing a cloud service or a requirement of a cloud service at a particular time. The message policy of an agent is the behavior which determines how many agents to send call-for-proposal messages out of all the feasible contractors recorded in its SCT. A successful message policy for an agent is its message policy that achieves 100% success in composing a cloud service or a service requirement for a particular state of the cloud composition environment. While, the best message policy is the successful message policy of an agent that ranks first amongst its peers in the same environment at a particular time. The message position utility is the major utility attributed to the agent based on its messaging position (β). Note that beta (β) is the ratio of the position or rank of the successful message policy (n) of the Cloud agent relative to the best message policy, to the total number of successful message policies (μ) at a specific time. Hence, the message utility of a Cloud agent, U can be computed as follows:

$$U(\beta) = \begin{cases} (1-\beta)+\varphi, & 1 \geq \beta > 0,\ \varphi = \frac{1}{\mu},\ \beta = \frac{n}{\mu} \\ 0\ , & otherwise \end{cases} \quad \textbf{(10)}$$

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Figure 4 shows the relationship between the message utility (U) and β for 100 successful message policies (μ). For example, to calculate the total message utility of a successful cloud agent at $\beta = 0.09$. We first compute the agents' base utility as $\varphi = \frac{1}{100} = 0.01$.The message position utility is $1 - 0.09 = 0.91$. Therefore, the total message utility is the sum of both utility components and is equal to 0.92. It can be deduced graphically from Fig.4 that agents need to minimize β in order to improve their message utility. This means agents should search for the best positioned successful message policy (i.e.,$n = 1$) for the current cloud composition environment.

As a result, cloud agents may gradually contract fewer or more contractors as the case may be in their SCTs for cloud service composition and could possibly reduce the number of messages exchanged while maintaining the percentage success of service composition.

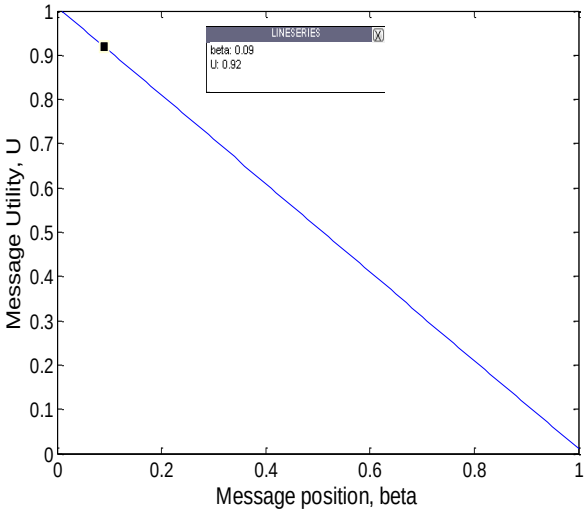

Fig. 4. Message utility of successful Cloud Agents for 100 successful message policies (μ=100).

V. AGENT BASED TESTBED

An Agent based testbed for minimizing messages propagated by cloud agents in agent-based cloud service composition was implemented using NetLogo (An Agent-based Simulation environment). The testbed comprised Web Services, RAs and swarms of CAs, BAs and SPAs with uniformly distributed message utilities (a value in [0, 1]). A Cloud agent with the best message policy has the best message utility in its environment (global and personal). In addition, the agent-based testbed was configured to adopt the semi-recursive contract net protocol with cloud composition agents searching for a satisfactory or most desirable successful message policy from the service capability tables at their disposal.

A set of experiments were carried out, using Particle swarm optimization (PSO) algorithm for evolving the optimal personal best messaging utility and global best messaging utility of Cloud agents. Each experiment was conducted under the assumption that for each swarm of agents there was only one true global optimal behavior.

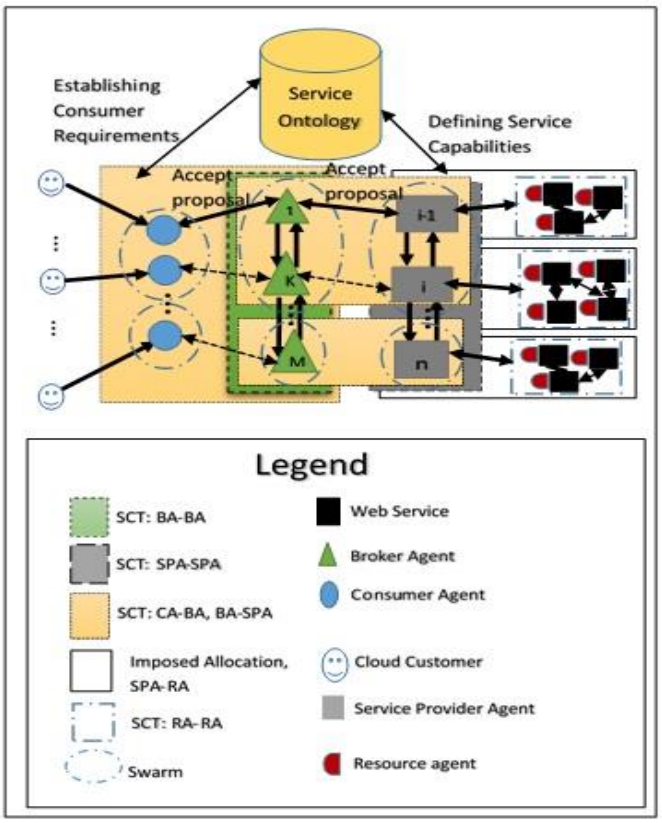

Fig. 5. Functional Architecture of the Agent-based testbed for Cloud service composition.

VI. EMPIRICAL RESULTS

In order to evaluate the ability of Cloud agents to minimize messages exchanged during cloud service composition, a set of experiments were performed using the agent-based testbed functional architecture depicted in Fig.5 such that all agents adopted SR-CNP equipped with SCTs. This testbed enabled investigation of the effect of having Cloud agents adopt an evolutionary search technique (PSO) for evolving their optimal personal best messaging utility and global best messaging utility. In the experiments, the cloud agents programmed with strongly connected SCTs and the RAs probability of failure was kept constant. Two performance measures were used to evaluate cloud agents. 1) The average message utility and 2) The average number of messages exchanged. Whereas, recording the average message utility for a constant RA probability failure provides a means to measure the effectiveness of using Cloud agents in agent-based cloud service composition, documenting the average number of messages exchanged provides a window to measure the efficiency of Cloud agents in minimizing the message overhead. The empirical results obtained from the experimental testbed of the preceding section are shown in Figs. 6 and 7. The parameters of the PSO algorithm used in this testbed are presented in Table 1.

Table. I. PSO parameters

PSO Parameter	Value
Population size/swarm	[50,100]
Particle-Inertia	0.98
Attraction to personal best Message Utility	2.0
Attraction to global best Message Utility	2.0

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

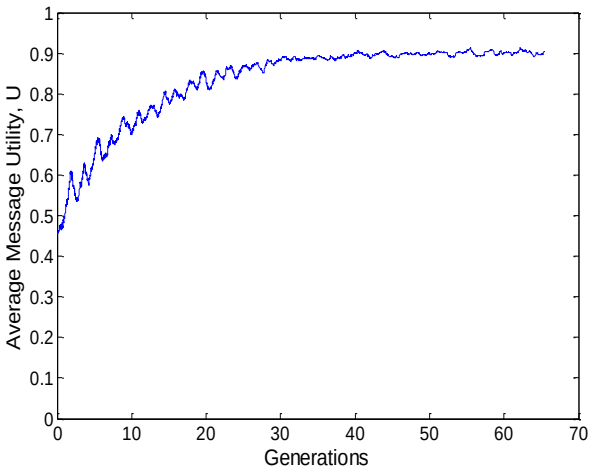

Fig. 6. Average Message utility of Cloud Composition agents.

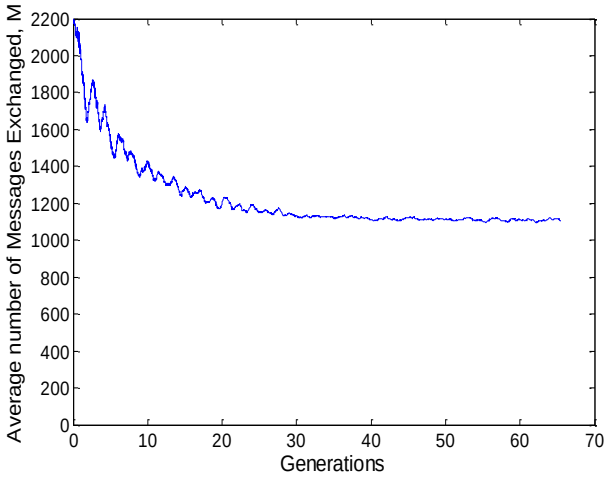

Fig. 7. Average Number of Messages Exchanged by Cloud Composition Agents.

A. Observation

Figures 6 and 7 show the average message utility and the average number of messages exchanged by Cloud agents. It is observed from these figures that for cloud agents in the agent-based testbed, the average message utility and average number of messages exchanged were maximized and minimized, respectively. More specifically, the results in Figs. 6 and 7 show that the average message utility achieved by Cloud agents is 0.91 and the average number of messages exchanged is 1095 messages.

B. Analysis

Cloud agents were able to maximize their message utility and minimize the number of messages exchanged because by using their knowledge (personal best message utility) and the knowledge of their immediate environment (global best message utility), they were able to improve their message utility by approaching the best message policy known to them. Accordingly, Cloud agents also minimized (by approximately 50%) the aggregate message exchanged.

VII. CONCLUSION AND FUTURE WORK

This paper has introduced a message efficient agent-based testbed for composing cloud services. Using SR-CNP and SCTs, cloud composition agents were able to adjust their message utilities in order to achieve optimal (best message utility) outcomes. The results in Figs.6 and 7 show that cloud agents can collectively improve their message utility and reduce the number of messages exchanged, respectively while successfully composing cloud services.

The contributions of this work include: 1) Mathematical analyses of message exchanges in agent-based cloud service composition; 2) Development of an Agent-based testbed comprising learning agents in agent-based cloud service composition; and 3) conducting experiments and obtaining empirical results to show that cloud agents can successfully minimize the number of messages exchanged while composing cloud services.

Finally, this work has reported observations and results from preliminary experiments. A more comprehensive set of experimental results and extensive discussions of observations will be presented in a future paper.

REFERENCES

[1] J.O. Gutierrez-Garcia and K.M. Sim, "*Agents-based cloud service composition*". *The international Journal of Artificial Intelligence, Neural Networks and Complex Problem-solving Technologies* 2012, 22(2).

[2] Windows Azure: Microsoft's Cloud platform, 2013. Available: http://www.windowsazure.com/

[3] What's new with Amazon Web Services, 2013. Available: http://www.aws.amazon.com/

[4] Over 3 million apps deployed to Google Cloud Platform, 2013. Available: https://cloud.google.com/

[5] Elastic Infrastructure at Your Fingertips from GoGrid, 2013. Available: http://www.gogrid.com/

[6] IBM Integrated Service Management for Cloud Service Provider, 2013. Available: http://ibm.com/http/www-01.ibm.com/software/tivoli/cloudcomputing/service-provider-platform/

[7] K. M. Sim, "Agent-based cloud computing". IEEE Transactions on Services Computing. October-December 2012, 5(4):pp564-577.

[8] J.O. Gutierrez-Garcia and K.M. Sim. Self-Organizing for Service Composition in Cloud Computing, Proc. 2nd IEEE int. conf. on Cloud Computing Technology and Science, Indianapolis, IN, USA, 2010, pp. 59-66.

[9] J.O. Gutierrez-Garcia and K.M. Sim, Agent-based Service Composition in Cloud Computing. Proc. 2010 Conf. on Grid and Distributed Computing, Dec. 13-15, 2010, Jeju Island, Korea.

[10] K. M. Sim, *"Cloud Intelligence: Agents within an InterCloud". Awareness Magazine*. The official magazine for Future and Emerging Technologies Proactive Initiative, funded by the European Commission under FP7.Available: http://www.awareness-mag.eu/pdf/005153/005153.pdf

[11] B. Bauer, J.P. Muller and J. Odell,"*Agent uml: a formalism for specifying multiagent software systems*", *Int J softw Eng. Knowl Eng* 11(3), 2001: pp 207-230.

[12] M. N. Huns and L.M. Stephens, Multiagent Systems and Societies of Agents (II), Lecture notes (2002), CSCE976.

[13] FIPA Contract Net Interaction Protocol Specification. *Foundation for Intelligent Physical Agents*, 2002. Available: http://www.fipa.org/specs/fipa00029/SC00029H.html.

[14] J. Kennedy and R.C. Eberhart, Swarm Intelligence. Morgan Kaufmann. ISBN 1-55860-5959.

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)