A Probabilistic Process Learning Approach for Service Composition in Cloud Networks

Ismaeel Al Ridhawi, Yehia Kotb
College of Engineering and Technology
American University of the Middle East (AUM)
Egaila, Kuwait
{Ismaeel.Al-Ridhawi; Yehia.Kotb}@aum.edu.kw

Moayad Aloqaily, Burak Kantarci
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Ontario, Canada
{maloqaily; burak.kantarci}@uottawa.ca

***Abstract*—We present a formal probabilistic framework for process learning to compose service specific overlays (SSO) in cloud networks. The approach provides a learning mechanism that relies on previous composition results to build service composition process models that can be adopted for future composition requests. The process is then translated into a workflow-net to provide guaranteed delivery of requested cloud media services to clients. A mathematical merge technique is also presented to converge multiple process threads into a single composed process. We provide simulation results to show that our approach can adequately establish sound composition paths in a timely manner.**

Keywords—Workflow-net, Petri-net, service composition, overlay networks, cloud networks.

I. INTRODUCTION

Nowadays, in order to cope with the boundless service preferences of cloud subscribers, current approaches such as centralized cloud service provisioning that incorporates increased amounts of resources at the datacenter is becoming ineffective in terms of scalability, cost and flexibility. With the aid of network-side devices, distributed cloud service availability provides an enhanced alternative. More specifically, service subscriber devices can be used as service adaptation nodes to help in the process of providing user-tailored composite cloud services.

The Service Oriented Architecture (SOA) has become a standard for service composition solutions [1]. With today's mobile cloud advancements and rapid deployment of roaming devices, the SOA paradigm has begun its shift to wireless networks. Most service composition solutions do not take into consideration mobility issues [2]. The seamless composition of distributed service components into more complex ones in mobile environments is a sensitive process. Issues such as service disruption caused by movement of service providers, heterogeneity of devices and resource variability all are examples of how unpredictable a mobile cloud environment can be.

Overlays provide a layer of abstraction and thus made it possible to use the lower-level network infrastructure to provide higher-level services to users. With the aid of mobile cloud service subscribers, service-specific overlays can be constructed to provide composable services such as user-tailored media content to help cloud application providers achieve QoS guarantees. Nonetheless, service-specific overlay composition in a dynamic mobile environment is a challenging contemporary issue for cloud management systems.

In this paper, we define a probabilistic framework for process learning using previous SSO composition node log files. Service composition logs or history is used to enhance system management issues such as learning and network optimization. This continuous feedback about the process execution and its impact on the network performance is very essential for cloud service providers. Previous composition results are used to build service composition process models that can be adopted for similar future composition requests. Process models are translated into workflow-nets to provide guaranteed delivery of cloud services to clients. We define process threads and a merge technique to combine multiple process threads that are part of the same process into a single composed composition process.

The paper is organized as follows. Section II discusses related literature review. Section III provides a model for the composition problem. Section IV introduces the probabilistic process learning approach. A representation of the learnt process using Workflow-nets is provided in Section V. Section VI provides some simulation results. Finally, Section VII concludes the paper and discusses some ongoing future work.

II. RELATED WORK

Developing efficient solutions to achieve automated service composition has been a hot topic since the introduction of web services. Since composite services are usually requested in a dynamic environment, adapting solutions applied on different network environments leads to inadequate composition results. Reinforcement learning is one commonly adopted technique used to achieve automation for service composition [3]-[6].

Casser et al. [7] presented a method using a probabilistic machine learning technique that processes service request templates or keyword queries and retrieves the most relevant services to the submitted request. The solution extracts latent factors from semantically enriched service descriptions which are then used to construct a model to represent different types of service descriptions in a vector form. A vector distance model is used to calculate the similarity of vector representations in the latent factor space. The similarity value is then used as a notion for similarity ranking. Although the

solution adopts a learning mechanism when composing services, the work is not intended for service overlay node composition.

The authors in [8] introduced a service specific overlay composition method which considers semantic similarity and semantic nearness between overlay nodes. A decentralized solution is adopted such that each overlay node will determine how semantically similar it is to the other mobile nodes in a dynamic network environment. Semantic similarity refers to how identical a node's service description is in relation to the requested service task description. According to the similarity score provided by a node, a decision is made to either accept or reject a service node from the overlay composition path. The solution does not consider a learning mechanism when creating composition paths, rather, the solution reiterates every time a node joins or leaves the network.

Despite the abundant research involved in applying reinforcement learning techniques for composite service adaptability, there is still limited research involved that aims to provide adaptable service specific overlays for today's cloud networks. This paper proposes a probabilistic learning technique used to create service composition processes which are adapted to newly introduced composite service requests in the cloud.

III. Modeling the Composition Problem

A. Preliminaries

We model the service overlay composition problem as a set of processes. A process is a series of actions performed to achieve a task. We assume that each action is performed by service nodes. For instance, assume that a media content exists at one of the mobile nodes in the cloud in which we call Media Server (MS). A request from the Media Client (MC) is received for that particular media content with some modifications to the original content, such as the addition of subtitles, encoding conversion, or size compression. In order to deliver the requested service, a set of actions must be performed to add the subtitles or compress the size of the content. To do so, those actions must be performed in sequence by other nearby mobile service nodes which we call Media Ports (MP). The set of mobile nodes involved in the service composition process from the MS to the MC form a SSO.

Figure 1 depicts an example of a service specific overlay such that MS provides the original media contents, MP_1, MP_2, and MP_3 add media enhancements to the original content in sequence to provide and deliver the requested composite service to the MC. An action performed by MP_1 in this example would be to encode the original video to another format provided by the MS. A second example of an action would be, after the encoding format conversion, MP_2 adds subtitles. The same process repeats for the other service node MP_3 until the requested composite service is delivered to the MC.

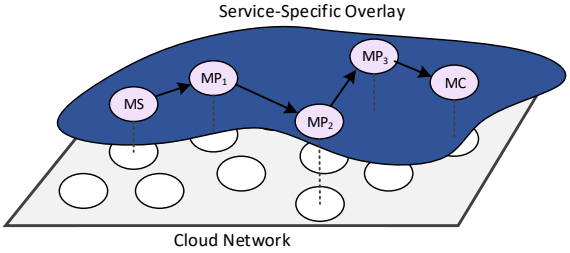

Fig.1. Service-specific overlay.

Events are the driving force behind any action. An example of an event is a movie that exists on the MS has been converted to another type of video encoding format. Provided this event an action can now be performed by an MP. We assume that there is a set $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_m\}$ of primitive events that cannot be fragmented into simpler events, where m is the number of distinct events in a process. We further classify Λ into two distinctive sets: Λ^0 and Λ^c. The former is the set of events in which a process begins with, such that $\Lambda^0 = \{\lambda_1^0, \lambda_2^0, \dots, \lambda_j^0\}$. The latter is the set of events that follow, such that $\Lambda^c = \{\lambda_1^c, \lambda_2^c, \dots, \lambda_k^c\}$. The properties for Λ are defined as follows:

$$\Lambda^0 \cup \Lambda^c = \Lambda \quad (1)$$

$$\Lambda^0 \cap \Lambda^c = \emptyset \quad (2)$$

$$\forall\, \lambda \in \Lambda^0, \bullet\lambda = \emptyset \quad (3)$$

$$\forall\, \lambda \in \Lambda^c, \bullet\lambda \neq \emptyset \quad (4)$$

where $\bullet\lambda$ is the set of earlier events that lead to λ. Given the definitions in (1) and (2), if j is the number of events in Λ^0 and k is the number of events in Λ^c, then $j + k = m$.

B. Workflow-Net

Processes are modelled using Workflow-nets which are an extension to Petri-nets. A Petri-net is a directed graph in which nodes are either transitions or places. A place is connected to one or more transitions. A transition is connected to one or more places. Places cannot be connected to places, and transitions cannot be connected to transitions. Thus, a transition must exist between places. The connections are made through directed arcs. Transitions perform actions and are represented as tokens residing in places. A transition is enabled only when there are no empty places (i.e. places with no tokens) connected to it as input. A transition executes (i.e. performs an action) after being enabled. The result of the execution is the removal of tokens from each of the transition's input places and the creation of tokens in each of its output places. Workflow-nets constitute an extension to Petri-nets such that they possess a single source and sink nodes. In other words, the Workflow-net possesses exactly one place with no incoming transition and exactly one place with no outgoing transition. This property achieves the notion of soundness which implies that the model is both structurally and behaviorally well-formed. Figure 2 depicts a Workflow-net model for the service composition process example outlined in Figure 1.

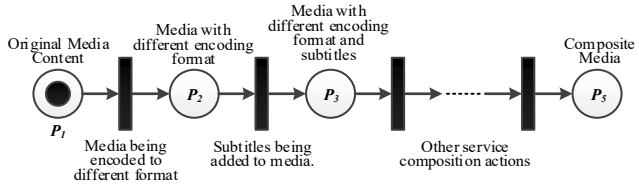

Fig.2. Service-specific overlay composition model using Workflow-net.

C. Definitions

The following are concepts used throughout this paper which are applied within the proposed composition solution.

Definition 1: a *pre-stage process source event* i for Λ^0 is used to ensure that regardless of the events that are happening, the process will always have a single event to start with. This aligns with the definition of a workflow-net in which the model contains a single place with no incoming transitions.

Definition 2: a *post-stage process sink event* o for Λ^c is used to ensure regardless of the flow of the process, it will always have a single event in which it terminates with. Again this aligns with the definition of a workflow-net in which the model contains a single place with no outgoing transitions.

Definition 3: an *activity* ξ_k is the lifetime of the media content k being composed inside a process, in which multiple actions are applied to produce the composed media content. An activity has to start with the source event i and end with the sink event o.

Definition 4: a *process thread* ϵ is a description of the process flow from source i to sink o, such that $\{\epsilon_1, \epsilon_u, \dots, \epsilon_u\}$ is the set of threads that build up a process.

Definition 5: a *dependency matrix* $\mathfrak{D}$ is a two dimensional matrix with length equal to the number of events defined in Λ. The matrix defines the dependencies between events as follows:

$$\mathfrak{D}[i,j] = \begin{cases} 0 & if\ \lambda_i = \lambda_j \mid \lambda_i\ does\ not\ depend\ on\ \lambda_j \\ 1 & if\ \lambda_i\ depends\ on\ \lambda_j \end{cases} \quad (5)$$

Definition 6: a *projection vector* ρ is used to project the dependency of a certain event with all events defined in Λ. Using a mask, only certain rows from the dependency matrix are shown while the others are hidden. For example $\rho = [0\ 0\ 1\ 0]$ is a vector used to study the dependency of λ_3 on four events defined in Λ.

Definition 6: a *projected event dependency* $\acute{\mathfrak{D}}$ is a vector which outlines the dependency of a single event from matrix $\mathfrak{D}$, and is derived as follows:

$$\acute{\mathfrak{D}} = \rho \times \mathfrak{D} \quad (6)$$

Definition 7: an *executed vector dependency* $\overrightarrow{\mathfrak{D}}$ determines the dependency of a certain event upon $\vec{\Lambda}$, which is the set of events that have already been executed. The vector is derived as follows:

$$\overrightarrow{\mathfrak{D}} = \vec{\Lambda} \times \mathfrak{D} \quad (7)$$

IV. Process Learning Approach

The composition process is based on a learning approach, such that node log files from previously successful composition processes are used to recreate similar compositions for similar events in the future. Service node log files include process threads' descriptions such as the events that lead to an action to be considered, the events which occurred after applying the action, list of nodes that preceded the considered node in the composition, and the node that followed the considered node in the composition.

Three steps are involved in the proposed service composition process learning method: *candidate events selection*, *probability of event occurrence*, and *converging threads into a process*. The first step derives a list of candidate events that may occur when applying a particular action. The second step calculates the probability that the selected candidate events will occur. The last step involves the convergence of threads into a process. A service composition process which may be made up of different or somewhat similar process threads (i.e. a series of events that occur following the actions performed by multiple service nodes) is formed by merging those threads. These steps are repeated for all process threads for newly composed service log files until all events are handled.

A. Candidate Event Selection

A set of events that are candidates for processing are selected in this step. The set of candidate events $\acute{\Lambda}$ are selected according to (8), such that it contains all events that could start a particular process and do not depend on any previous events Λ^0, union with the set of events that depend on the already fired events $(\overrightarrow{\mathfrak{D}} \times \Lambda^c)$.

$$\acute{\Lambda} = \Lambda^0 \cup (\overrightarrow{\mathfrak{D}} \times \Lambda^c) \quad (8)$$

Those candidate events provide an overview of the events' sequence in a service composition process beginning with the pre-stage process source event i and ending with the post-stage process sink event o. The set $\acute{\Lambda}$ contains all possible process threads' events that may exist in a single composition process and thus multiple event sequences are generated, which in turn form multiple threads.

B. Event Occurrence Probability

Once the set of candidate events $\acute{\Lambda}$ have been derived according to (8), the next step is to determine the probability for an event to occur. This is achieved by first calculating a belief value for an event occurrence according to (9).

$$bel(\lambda_i) = \int_{j=1}^{n} P(\lambda_i|\lambda_j) \times MAX(\overrightarrow{\mathfrak{D}}_j, \sigma(\|\lambda_i \cap \Lambda^0\|)) d_\lambda \quad (9)$$

where $P(\lambda_i|\lambda_j)$ is the probability for the event λ_i to occur given the occurrence of the event λ_j. This probability is calculated according to a uniform distribution function. For instance, if the occurrence of event λ_j attains three other possibilities of events which are $\lambda_{i1}, \lambda_{i2}$ and λ_{i3}, then $P(\lambda_{i1}|\lambda_j) = 1/3$. $\|\lambda_i \cap \Lambda^0\|$ is the magnitude of the intersection between event λ_i and the set of events in which a process begins with, σ is a step function which produces 0 if the magnitude is 0 and 1 otherwise.

Thereafter, once the believe values have been derived for all candidate events, a normalized probability is calculated for each event as follows:

$$P_N(\lambda_i) = \frac{bel(\lambda_i)}{\sum_{j=0}^{n} bel(\lambda_j)} \quad (10)$$

C. Converging Threads into a Process

As defined in Section III.C, process threads are the building blocks for a process. A thread provides a description of the events that are produced in a process following the actions

performed by service nodes. More specific towards media service composition, threads constitute the different node composition paths taken to provide the composite service. The previous two steps of the process learning method develops a composition path (thread) from the set of candidate events. In this step the process which may be composed of multiple threads is built by converging those threads together. Thread convergence is the operation of redefining the relationship between threads and merging them into a bigger entity.

We assume that thread events are defined through a sequence of levels in which events are categorized by their hierarchy level. Two different threads having the same set of events occurring at the same hierarchy level can be merged into the same process. Therefore, the operation of thread convergence is achieved through thread hierarchy level comparison. For instance, the example depicted in Figure 1 illustrates the nodes selected and the path considered to compose the requested service. The same requested service can also be composed through a different set of nodes and path (i.e. different overlay). Both solutions may create similar thread events that belong to the same execution sequence level. Given such circumstances, we can converge the two threads into a single process.

The steps involved in the convergence process are as follows:

1. Create a common start event λ_0 which is the starting event for all different composition paths (threads). E.g. availability of original non-enhanced movie on a node.
2. Create a common end event λ_{end} which is the final event for all different composition paths. E.g. movie enhancements completed.
3. $\forall\ (l_i \in \epsilon_n$ and $\forall\ l_j \in \epsilon_m)$, if $i = j$ then $\forall\ (\lambda_s \in l_i$ and $\lambda_t \in l_j), \lambda_c = \lambda_s \cup \lambda_t$.

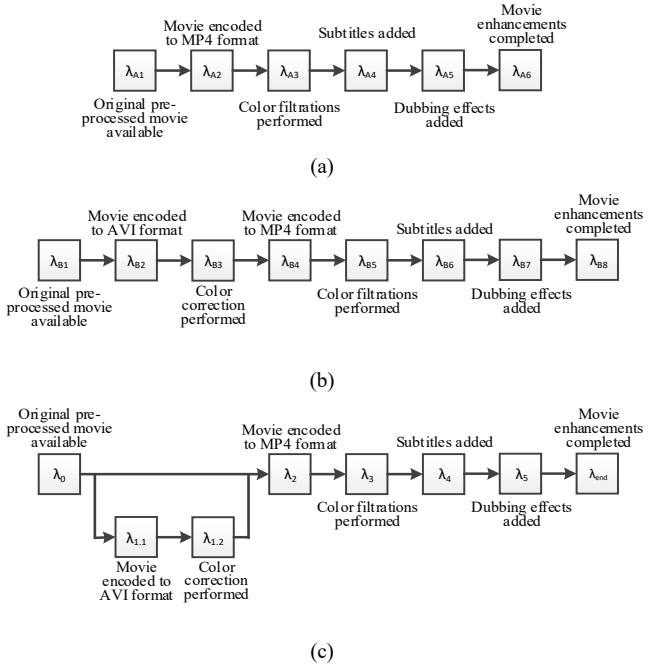

Fig.3. (a) Process thread 1 – events for service overlay composition path 1, (b) Process thread 2 - events for service overlay composition path 2, (c) Process convergence – merged threads for the same composition process.

4. $bel(\lambda_c) = \int P(\lambda_c) = \int P(\lambda_s) + P(\lambda_t) d_\lambda$.
5. After convergence is complete for all threads, obtain the normalized probabilities from the belief values found in step 4.

where l_i is a hierarchy level in thread ϵ_n and l_j is a hierarchy level in thread ϵ_m. The convergence process starts by creating a single starting event λ_0 and ending event λ_{end} that is common among all threads. Then, for each hierarchy level, the common events are modeled once with a probability of occurrence equal to the sum of the two occurrences before convergence. Finally, the probability is then normalized.

Figure 3 provides an illustrative example of two overlay composition paths (threads) considered for the addition of media enhancements requested by a MC to an original movie content found at a MS. Figure 3a illustrates the thread events involved in the composition process. Figure 3b illustrates different but similar thread events in the composition process. Figure 3c provides an illustration of the convergence of the two threads in which common events are modelled once only.

V. Workflow-net Representation of the Learnt Process

A. Node Selection

Once the process learning of previous composition tasks is complete, current composition requests would rely on those learnt processes to generate a new composition path that is adequate for the current network configuration. Therefore, when a service request from the current service node MP_i is transmitted to another node MP_{i+1}, the thread to be followed to achieve the requested process must be determined. The decision of which thread to follow is determined based on the task coverage achieved by employing the currently available nodes. Task coverage is defined as follows:

$$\forall \vartheta \in R, \mathcal{L}(\vartheta, \lambda) \neq \emptyset \text{ and } \lambda \in \Lambda \tag{11}$$

in other words, for all actions ϑ that form the requested service R, those actions must be part of the set of capabilities provided by the process. $\mathcal{L}$ is an association function that associates an action ϑ to an event λ occurrence.

If the node receiving the service request can achieve the requested action independently, that is:

$$\forall \vartheta \in R, \vartheta \in \varrho(MP_i, \vartheta) \tag{12}$$

then the node will no longer need to send a service composition request to another node, hence, the task is achieved independently. ϱ is a mapping function that maps capabilities to nodes. Otherwise, if a single node cannot provide the requested service, then the service must be composed with the aid of other service nodes. This is achieved using a capability matrix (13).

A capability matrix M outlines each node's capabilities:

$$M = \begin{array}{c} \\ MP_1 \\ MP_2 \\ \vdots \\ MP_k \end{array} \begin{array}{c} \begin{array}{cccc} \vartheta_1 & \vartheta_2 & \cdots & \vartheta_n \end{array} \\ \begin{bmatrix} 1 & 0 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{array} \tag{13}$$

where

$$M(MP_k, \vartheta) = \begin{cases} 1 & if\ node\ MP_i\ can\ perform\ action\ \vartheta_n \\ 0 & otherwise \end{cases} \quad (14)$$

The service node that receives the composition request will traverse the matrix until all required actions are selected for a particular composition to be attained. This traversal will feature the set of nodes considered and their capabilities used to compose the requested service.

B. Workflow-net Formulation

Once the set of nodes used to attain a process are determined, the workflow which describes the behavior of a node MP_i is developed and executed as follows:

$$\forall MP_i \in MP\ \exists\ wf_j | wf_j \in \epsilon_j \text{ and } \forall_{i=1}^{k} MP_i \cdot MP_{i+1} \quad (15)$$

such that the nodes $\forall_{i=1}^{k}\ MP_i \cdot MP_{i+1}$ execute the process thread ϵ_j. This can be represented as a set of Workflow-nets $\forall MP_i \in MP$, such that each Workflow-net is described as follows:

$$wf = \langle P, T, P{\times}T \cup T{\times}P \rangle, \text{ and} \quad (16)$$

$$\exists\ p_i \in P | {\bullet}p_i = \emptyset, \text{ and} \quad (17)$$

$$\exists\ p_o \in P | p_o{\bullet} = \emptyset \quad (18)$$

where P and T are places and transitions in the workflow respectively. Places are events and transitions perform actions. p_i and p_o are the input and output places to the workflow.

The complete service overlay composition framework $\mathcal{F}$ is described as follows:

$$\mathcal{F} = \langle WF, wf{\times}wf, \vartheta, R, MP, \varrho, \varsigma \rangle \quad (19)$$

where WF is the set of workflows, $wf{\times}wf$ describes the flow of data from one node to another that belong to the composition set, ϑ is the set of actions performed, R is the requested composite service, MP is the set of service nodes that are part of the service specific overlay, ϱ is a mapping function between a node and an action performed by that node, ς is a one to one mapping between a node MP_i and the workflow wf_j.

C. Soundness of the Framework

Soundness guarantees that the defined process will produce an output (i.e. composed media content) when given an input (i.e. original media content). In this section we introduce a theorem of soundness to define the conditions that need to be satisfied for a process to be sound. Additionally, proofs are provided to support the theorem.

Theorem 1: A framework $\mathcal{F}$ is sound if and only if:

1. $\forall r \in R, r \in \vartheta$.
2. $\forall wf \in WF, wf$ is sound.
3. $wf{\times}wf$ is a sound wofkflow.
4. At any time t, if a node MP_i leaves the network $\exists MP_j | \forall \vartheta \in \varrho(MP_j, \vartheta)$.

To proof the theory, we need to proof that if $\mathcal{F}$ is sound, then all four conditions are satisfied, and vice versa.

Proof 1: Since $\mathcal{F}$ is sound, therefore for every input place p_i there is a given output place p_o. Therefore $\forall r \in R\ \exists\ \vartheta | r \equiv \vartheta$. Since the input p_i exists in node MP_i and the output place p_o is in node MP_j, therefore $\forall wf \in WF$ such that $\varsigma(MP_i, wf_j)$ is true. Therefore wf is sound, and therefore $\forall wf_i, wf_j | wf_i{\times}wf_j \neq \emptyset, wf_i{\times}wf_j$ is also sound. Therefore $\forall \vartheta \in R$, the action ϑ satisfies the task coverage.

Proof 2: Since $\forall r \in R, r \equiv \vartheta$, therefore ϑ is self-sufficient. Since wf is sound and $wf{\times}wf$ is also sound, therefore the input place p_i will eventually reach the output place p_o. Since task coverage is always maintained, therefore $\mathcal{F}$ is sound.

VI. Simulation Results

To generalize the problem and test the system's capability regardless of the type of service requested, we developed a simulator to analyze three different solutions: i) the proposed learning-based node cooperation method, ii) a non-learning-based node cooperation method, and iii) a non-cooperative method. The first considers a solution which uses the probabilistic process learning approach in which overlay nodes cooperate to deliver the requested composite service. The second solution considers a similar cooperative model in which nodes cooperate to achieve the requested composite service but without any aid from the probabilistic learning module. The third solution disregards node cooperation and hence services are composed (if possible) using a single overlay node. The goal of these simulation tests is to empirically demonstrate that our definition of learning-based cooperation is correct and that process threads can be adequately established and carried out.

The input to the simulator consists of a thread in the form of a linear logic expression with operators described in [9]. Other input parameters consist of a set of nodes, each with a set of service capabilities, expressed as Workflow-nets. Each capability corresponds to one action defined in the process, along with the cost associated with performing that action. Actions that are not part of a node's set of capabilities have their cost set to infinity. A uniformly distributed random variable is used to first determine the initial set of capabilities for each node. When a node is assigned a service capability, the cost for executing the action is randomly determined with a normally distributed variable.

Once node capabilities are set, the simulator evaluates the task coverage. If the generated nodes' capabilities are insufficient to provide a complete task coverage, the simulator terminates with infinity as a cost for execution. Otherwise, the cooperative process is constructed and executed. For simplicity, the execution time is computed as the total execution cost in the Workflow-net.

The first experiment conducted considers the delay incurred to complete a certain number of service requests. The results depicted in Figure 4 show that the non-cooperative solution incurs the most delay in comparison with the two cooperative solutions. Although both cooperative solutions show that the time needed to complete the service requests stabilizes as the number of service requests increases, the probabilistic process learning approach outperforms the non-learning approach by almost 29%. This is due to the use of

composition paths which have been applied for previously considered composition requests.

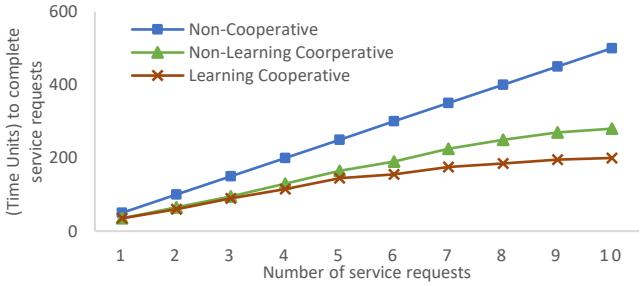

Fig.4. Time required to complete a number of service requests.

The second experiment considers a service request which requires a set of actions to be performed to achieve the task. Results depicted in Figure 5 show that the learning cooperative approach outperforms the non-learning cooperative and non-cooperative approaches by 17 and 61 time units respectively (i.e. 30% and 61% reduction in delay respectively).

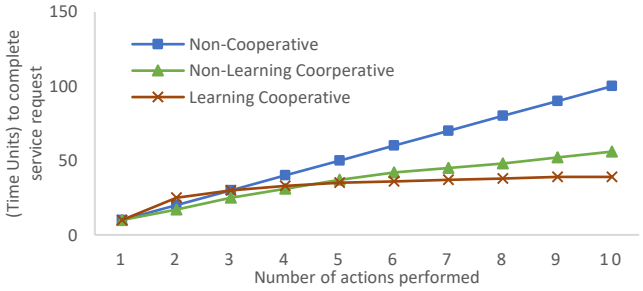

Fig.5. Time required to complete a single service request composed of multiple of actions.

The third experiment outlines the improvements achieved as the number of nodes used for cooperation increases. Results shown in Figure 6 prove that the learning approach adds an increased benefit as the number of nodes increase such that with the availability of 10 nodes to be used for cooperation, the delay is reduced by 54% in comparison to the non-learning approach.

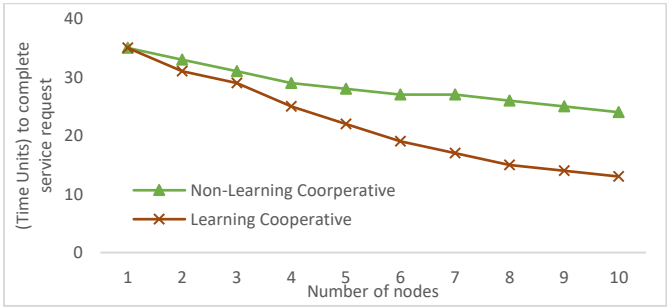

Fig.6. Time required to complete a single service request composed of multiple actions as the number of nodes used for cooperation is varied.

Another simulation was conducted to test the stability of the framework against node departure. Results depicted in Figure 7 show that the overhead incurred to create a cooperative composition solution increases almost exponentially for the non-learning cooperative approach. On the contrary, although the overhead also increases for the learning cooperative approach, the delay incurred for creating a cooperative composition is much lower than the non-learning approach.

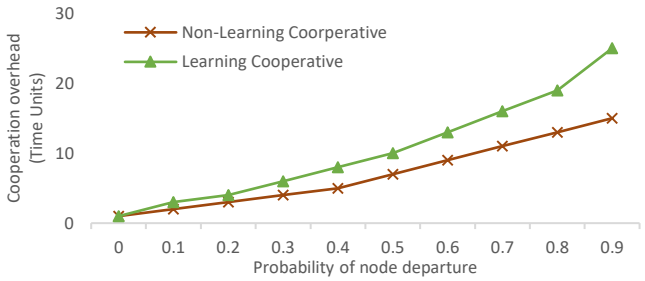

Fig.7. Overhead incurred as the probability of node departure increases.

VII. Conclusion

This paper introduced a probabilistic learning technique used to create service composition processes which are adapted to newly introduced composite service requests in the cloud. Service composition logs are used to enhance system management issues in which continuous feedback about the process execution and its impact on the network performance are used to build service composition process models that can be adopted for similar future composition requests. Process models are translated into workflow-nets to provide guaranteed delivery of cloud services to clients. Simulations were conducted to compare the proposed solution to two other service composition techniques.

References

[1] H. Petritsch. “Introduction to SOA and Web Services” in Integrating SOA and Web Services, River Publishers, Denmark: River Publishers, pp.9-10, 2011.

[2] S. Deng, L. Huang, J. Taheri, J. Yin, M. Zhou, A. Y. Zomaya, "Mobility-Aware Service Composition in Mobile Communities," in IEEE Transactions on Systems, Man, and Cybernetics: Systems , vol. no.99, pp.1-14.

[3] H. Wang, G. Huang and Q. Yu, "Automatic Hierarchical Reinforcement Learning for Efficient Large-Scale Service Composition," in Proc. 2016 IEEE International Conference on Web Services (ICWS), San Francisco, CA, 2016, pp. 57-64.

[4] Y. Lei, Z. Jiantao, W. Fengqi, G. Yongqiang and Y. Bo, "Web Service Composition Based on Reinforcement Learning," in Proc. 2015 IEEE International Conference on Web Services, New York, NY, 2015, pp. 731-734.

[5] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng and A. Bouguettaya, "Adaptive and Dynamic Service Composition via Multi-agent Reinforcement Learning," in Proc. 2014 IEEE International Conference on Web Services, Anchorage, AK, 2014, pp. 447-454.

[6] L. Yu, W. Zhili, M. Lingli, W. Jiang, L. Meng and Q. Xue-song, "Adaptive Web Services Composition Using Q-Learning in Cloud," in Proc. 2013 IEEE Ninth World Congress on Services, Santa Clara, CA, 2013, pp. 393-396.

[7] G. Cassar, P. Barnaghi and K. Moessner, "Probabilistic Matchmaking Methods for Automated Service Discovery," in IEEE Transactions on Services Computing, vol. 7, no. 4, pp. 654-666, Oct.-Dec. 2014.

[8] Y. A. Ridhawi and A. Karmouch, "Decentralized Plan-Free Semantic-Based Service Composition in Mobile Networks," in IEEE Transactions on Services Computing, vol. 8, no. 1, pp. 17-31, Jan.-Feb. 2015.

[9] Y. T. Kotb, S. S. Beauchemin and J. L. Barron, "Workflow Nets for Multiagent Cooperation," in IEEE Transactions on Automation Science and Engineering, vol. 9, no. 1, pp. 198-203, Jan. 2012.