MJCIS Vol. 13 No.1 Dec2017

Mansoura Journal of Computers and Information Sciences

Automatic Cloud-Based IoT Mashup Algorithm

Dalia Elwi
Faculty of computers and information systems , C.S dep.
Mansoura University, Egypt
dalia_elwi@yahoo.com

Omaima Nomair
Faculty of computers and information systems , C.S dep.
Mansoura University, Egypt
omnomir@yahoo.com

Samir Elmougy
Faculty of computers and information systems , C.S dep.
Mansoura University, Egypt
samirelmougy@yahoo.com

ABSTRACT

Internet of Things (IoT) and cloud computing are two of the most important trends in information and communication technology that attract the attention of many researchers in recent years. A new trend is raised from integrating both trends called Cloud of Things (CoT). In this paper, we focus on integrating IoT with cloud computing because of the benefits that IoT can gained from unlimited storage and unlimited processing capabilities provided by cloud computing. Firstly, we propose a CoT architecture that supports Things as a Service (TaaS) and IoT Mashup as a Service (MaaS). Secondly, we develop an automatic IoT Mashup Algorithm (IoTMA) for application development in less response time by composing existing things services and web services without needing of high experience in programming. Experimental results proved that our algorithm reduced the response time compared to some other recent related works.

General Terms

Internet of Things, Cloud Computing, Cloud of Things, Mashup.

Keywords

Internet of Things, Cloud Computing, Cloud of Things, Mashup, CoT Architecture, IoTMA algorithm, Planning Graph,

1. INTRODUCTION

The Internet of Things (IoT) [1] is a technology where all things in the world such as devices, humans and animals are connected to the internet and have a unique address URL, which makes the communication easier. So, IoT things can be monitored and controlled remotely. The number of things connected to the internet is increasing day after day dramatically. Therefore, Internet Protocol version 6 (IPv6) is implemented to solve the huge increasing problem. IoT is used in many applications to facilitate our lives [17] such as transportation, medical healthcare, agriculture, smart cities, and monitoring systems.

Cloud computing [18] is a computing technique used to deliver computing resources as a services over the internet such as servers, storage, databases, software…etc. The company that hosts their services in the cloud is called the service provider. Service consumer pays for the services and resources he/she is used (Pay-as-you-go). There are three types of cloud computing Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [18, 19]. IaaS offers all infrastructure services like servers, storage, virtual machines, and networks resources for renting rather than purchasing. PaaS offers software developing, testing, delivering, and managing services needed by developers. SaaS offers on-demand software applications over internet without installation requirements.

IoT challenges such as limited storage, and limited computational capabilities could be solved by integrating IOT with cloud computing in which the integrated trend is called Cloud of Things (CoT). In CoT, cloud computing has virtually unlimited capabilities and provides everything as a service. Also, it provides Things as a Service (TaaS) beside SaaS, PaaS, and IaaS.

Services Mashup is a web application used to create a new service from existing services composition in consistent state with the new requirements. For example, museum branches' addresses and pictures may be tied with a Google map for establishing a map mashup [3]. Mashup approaches can be classified based on several criteria such as manual vs automated approach, and single-source vs multiple-sources.

In manual approach, the user should create a mashup by using either composition language such as Business Process Execution Language (BPEL) or drag and drop GUI. This approach is time consuming, error-prone, and requires the user to have a great knowledge about services needed for mashup. However, in automatic approach the user makes a request then services discovery, selection and composition are automatically executing. This approach requires the user to specify the request precisely. There is another mashup approach called semi-automated approach that aims to assist the user at each step of the services composition procedure [4].

Single-source mashup means that each service can be composed with only one other so the mashup is considered as a single source path. However, multiple-sources means that each service can be composed with more than one other so the mashup is considered as a tree not as a path.

In this work, we propose a CoT architecture consists of four types of services: SaaS, PaaS, IaaS, and MaaS. Also, it supports things as a service where they can be accessed based on a service-oriented model, and mashup as a service with developing a related automatic IoT Mashup Algorithm (IoTMA) for IoT applications development. Where, IoTMA is modified from service composition algorithms over the graph plan discussed in [9] to extract the optimal solution in less response time.

In Section 2, we discuss some related works. In Section 3, we present our proposed CoT Architecture and IoTMA algorithm. We present the experimental results compared with other mashup algorithms results in Section 4. In Section 5, we conclude our work.

2. RELATED WORKS

There are several researches had been done in integrating IoT and cloud computing. In [10], Zhou et al. analyzed IoT requirements for smart home application and presented a cloud architecture for dynamic service composition. In [11, 12], Distefano et al. assumed that things can be accessed according to service oriented architecture and therefore they employed a cloud paradigm for enabling sensing and actuation as a service. In [13], Janggwan et al. proposed a model for IoT mashup called IoTMaaS based on the model driven architecture and cloud computing. In addition, they designed a cloud based platform to execute their model and implemented a prototype platform for proving the architecture concepts. In [14], Bhattasali et al. focused on the integration between IoT and Cloud computing from the point of view of security where they proposed secure trusted things as a service based on encryption approach and a trust model. In [15], Blackstock and Lea developed a toolkit for manual IoT mashup called WoTkit in which it is a java web application includes user dashboard to visualize sensor data and mashup processing. In [16], Kleinfeld et al. built a manual IoT composition platform called glue.things by integrating and adapting a popular open source solution addressed in IoT domain. Glue.things is presented as a web of things hub for everyday things in our lives. In [9], Yan and Chen proposed a new QoSGraphPlan algorithm for multiple-sources web service mashup, which return correct solutions but it takes a long time and with redundant web services.

3. THE PROSOSED CoT ARCHITECTURE AND IoTMA ALGORITHM

In Section 3.1, we present our proposed CoT architecture. In Section 3.2, and Section 3.3, we explain some definitions of IoT mashup and planning graph based on [20], [21], and [9]. In Section 3.4, we discuss our proposed algorithm (IoTMA).

3.1 CoT Architecture

IoT can take advantage of unlimited capabilities of cloud computing in storage and processing data to improve the performance of its applications. Therefore, we propose a CoT architecture as shown in Figure 1. Where, there are four layers: a) IaaS layer for virtualization and hardware provided by the Cloud, b) PaaS layer includes databases and APIs for web services, and IoT things that can be used to access their data, c) MaaS as a new layer that contains a set of services for IoT mashup which discussed in details in the next section, and d) SaaS layer for software applications that is provided by the cloud.

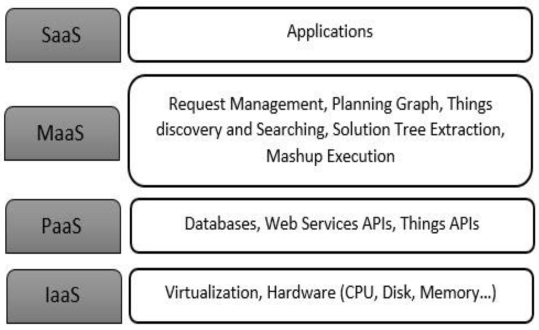

SaaS
Applications
MaaS
Request Management, Planning Graph, Things discovery and Searching, Solution Tree Extraction, Mashup Execution
PaaS
Databases, Web Services APIs, Things APIs
IaaS
Virtualization, Hardware (CPU, Disk, Memory...)

Figure 1. The proposed CoT architecture

MaaS layer is responsible for automatic IoT mashup to accelerate IoT application development without high programming skills. A shown in Figure 2, MaaS layer consists of:

- Request management service for user request analysis and for preparing a new service with user specification.
- Create planning graph by searching for the suitable IoT things and web services that can be mashed up to get the desired output from the available user inputs and conform to the user specifications.
- Solution tree extraction service extracts the optimal solution from the graph plan among all solutions constructed by the previous service.
- Mashup execution service executes the solution by accessing all APIs contained in the solution tree to get the needed data to be composed according to the desired output.

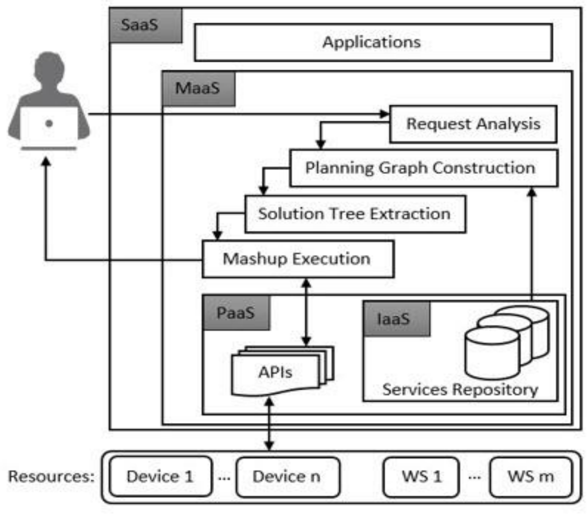


Figure 2. MaaS layer services for IoT mashup

3.2 IoT Mashup

IoT thing is a device connected to the internet that has a unique identifier (URL) to access that device in order to get its data. Therefore, it can be dealt with it like the Restful web services [2] through HTTP request.

Mashup is enabled when there are a set of services (IoT services and Web Services) that can be integrated so that they conform to the user request.

Definition 1: Let user request consists of *S* available inputs parameters and *T* desired output parameters in which each service has input parameters, output parameters, and Response Time (*RT)* as presented in Table 1.

Table 1. A set of IoT Services and Web Services

Services	Inputs	Outputs	Response Time (RT)
S1	a	e	100
S2	b, c	f, j	200
S3	e, f	h	400
S4	J	f, z	100
S5	z	d, h	200
S6	f	d	100

IoTMA is enabled if and only if:

- There is a service X where $X.inputs \sqsubseteq S$.
- There is a service Y where $T \sqsubseteq Y.outputs$.
- There are a set of *Mashupable Services* that can be integrated with X and Y as shown in Figure 3.

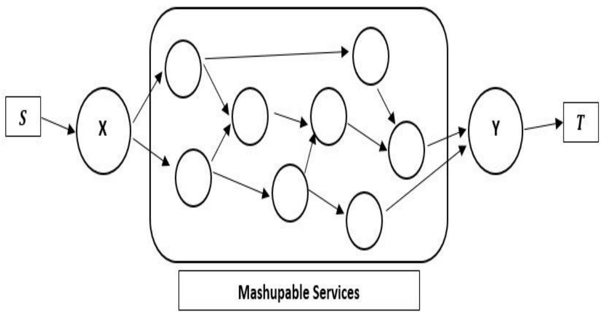


Figure 3. Enabled Mashup from S to T

Definition 2: The two services $s1$ and $s2$ are *Mashupable Services* if and only if:

- $s1.inputs \sqsubseteq s2.outputs$ where $s2$ is called *Producer service* while $s1$ is called *Consumer service* as shown in Figure 4 where *i*, *e*, *f*, *h*, *e*, *r*, and *o* are parameters.

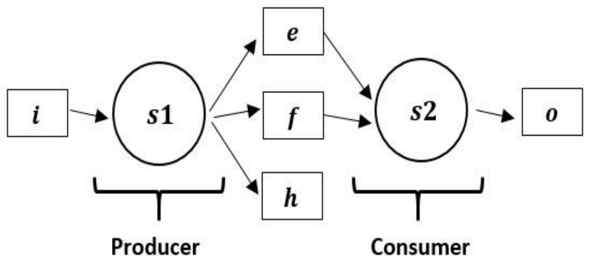


Figure 4. Mashupable Services

3.3 Planning Graph Technique and Solution Extraction

The proposed automatic IoT Mashup algorithm depends on using AI Planning Graph technique [7] for mashup planning with applying Dijkstra algorithm [8] for solution extraction.

Graph plan is used for mashup planning as a set of A layers and P layers, where A and P represent service and parameter layers respectively as shown in Figure 5.

Definition 3: Let $G = \{P0, A1, P1, \ldots, An., Pn\}$ is a graph where:

- $P0$ is an initial layer contains S parameters.
- An layer is a set of services $\{s_1, s_2, \ldots, s_m\}$ in which each service s has input parameters called *Preconditions* defined as $Pre(s) \sqsubseteq Pn - 1$.
- Pn layer is a set of parameters $\{p_1, p_2, \ldots, p_i\}$ in which $\forall\, p\, \forall\, s \in An-1, p \in s.outputs$. In this case s is called $Parent$.
- Last layer in G is called $lastP$.
- The solution is found if and only if the T parameters $\subseteq$ $lastP$ layer.

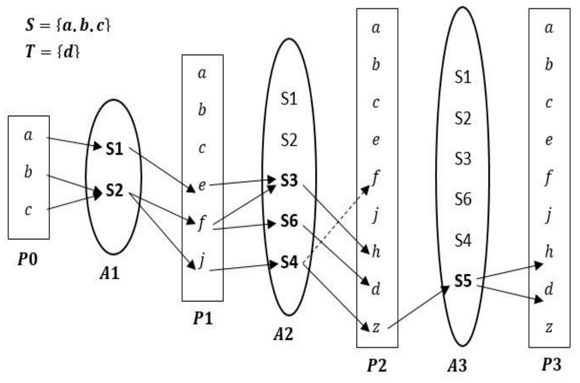


Figure 5. Mashup Graph Plan

Definition 4: $\forall$ parameter $p \in P$ layers, P may have more than one parent so that the *Best Parent* of p is a service s with the minimum cost where:

- $s \in p.Parents$.
- $Cost(s) = s.RT + Max(Cost(Pre(s)))$ where $\forall\ p \in Pre(s)$,
  $Cost(p) = Min(Cost(p.Parents)) = Cost(p.BestParent)$.

To extract the solution tree, backward chaining version of Dijkstra algorithm [8] is applied from all T parameters to any S parameters. The solution extraction is depending on *Best Parents* of each p from $lastP$ layer to $P1$ layer and *Preconditions* of each s from $lastA$ layer to $A1$ layer as in Figure 6.

Definition 5: Solution tree is extracted by Dijkstra backward chaining from *T* to *S* where:

- Step 1: $\forall\ T$ parameters $\in lastP$ layer, extract *Best Parents* set denoted by β.
- Step 2: $\forall\ s \in \beta$, extract *Preconditions* set denoted by γ.
- Step 3: $\forall$ parameter $p \in \gamma$, extract *Best Parents* set denoted by $\beta 1$.
- Step 4: Repeat steps 2 till 3 until reaching $\gamma \sqsubseteq S$.

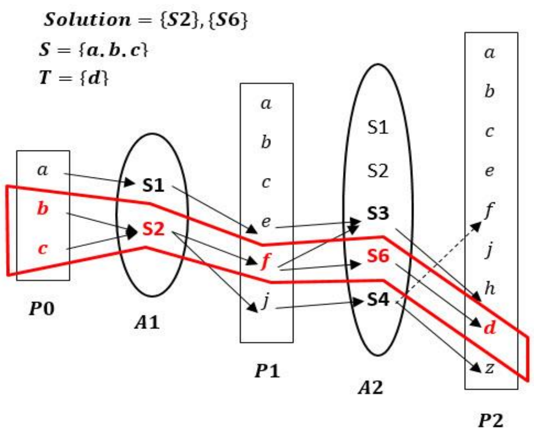


Figure 6. Solution Extraction

3.4 The Proposed IoTMA Algorithm

The proposed IoTMA is automatic algorithm modified from service composition algorithm over the graph plan [9] to extract the optimal solution. The criteria of optimal solution extracted by our algorithm are:

- Correct solution (get T from S).
- Solution has minimum cost.
- Solution is a tree, which means multiple source solution may be found.

Algorithm 1: IoTMA Algorithm
Data: S parameters, T parameters.

```
1.  Let global number called $CurrentGoalCost = \infty$.
2.  Let global list called $SolutionList = \emptyset$.
3.  $Mashup(S,T)$.
4.  $PrintSolution(SolutionList)$.
```

Algorithm 2: Mashup Algorithm
Data: S parameters, T parameters.

```
1.  Initialize planning graph $G$ with empty layers.
2.  Initialize $P0$ layer with $S$ parameters.
3.  Add $P0$ to $G$.
4.  Repeat
5.      $G = ExpandGraph(G)$.
6.      If $T \subseteq Pn$ layer then
7.          $ExtractSolution(G,T)$.
8.          $G = UpdateGraph(Pn)$.
9.  Until $StopPoint(Pn, Pn-1)$.
```

Algorithm 3: ExpandGraph Algorithm
Data: $G = \{P0.A1.P1.\ldots.An.Pn\}$

```
1.  $An = \{(s,cost) \mid Pre(s) \sqsubseteq Pn-1, Cost(s) = s.RT\}$
    .
2.  $Pn = Pn - 1$.
3.  For each $s \in An$ do
4.      For each parameter $p \in s.outputs$ do
5.          $Cost(p) = s.RT + Max(Cost(Pre(s)))$.
6.          If $p \in Pn$ then
7.              If $Cost(p) < p.OldCost$ then
8.                  Remove $old\ p$.
9.                  Add $new\ p$ with min cost to $Pn$.
10.         Otherwise
11.             Add $p$ to $Pn$.
12. Add $An, Pn$ to $G$.
13. Return $G$.
```

Algorithm 4: ExtractSolution Algorithm
Data: G graph, T parameters.

```
1.  Let $GoalCost = Max(Cost(p))\ \forall\ p \in T$.
2.  If $GoalCost < CurrentGoalCost$ then
3.      $CurrentGoalCost = GoalCost$.
4.      Set $SoutionList = \emptyset$.
5.      $FindSolution(G,T)$.
```

Algorithm 5: FindSolution Algorithm
Data: G graph, T parameters.

```
1.  For each $A$ layer in $G$ starting from $An$ to $A1$ do
2.      Select Best Parents set
    $\beta = \{ s \mid s\ is\ p.BestParent, \forall\ p \in T\}$.
3.      Add $\beta$ to $SolutionList$.
4.      $T = \{Pre(s), \forall\ s \in \beta\ \}$.
```

Algorithm 6: UpdateGraph Algorithm
Data: Pn layer.

```
1.  For each $p \in Pn$ do
2.      If $Cost(p) \geq CurrentGoalCost$ then
3.          Remove $p$ from $Pn$.
```

Algorithm 7: StopPoint Algorithm
Data: Pn layer, $Pn-1$ layer.

```
1.  If $Pn = Pn-1$ then
2.      Return $true$.
3.  Otherwise
4.      Return $false$.
```

Algorithm 8: PrintSolution Algorithm

```
1.  For each $Best\ Parents$ set $\beta \in SolutionList$ do
2.      Print $s$, $\forall s \in \beta$.
```

Algorithm 1 initiates the global variable named CurrentGoalCost with ∞ and empty global list called SolutionList, then executes a mashup process using S, T parameters and the existing services and finally prints the solution. Algorithm 2 is responsible for the mashup process where it firstly initiates the graph G with only one layer called P0, which contains S parameters (steps from 1 to 3). Then, expands G and checks if the desired output is appeared then extract the solution and update the graph (steps from 5 to 8). After that repeats steps from 5 to 8 until the stop point is reached. Algorithm 3 expands G by calculating the next A, P layers. A_n is a set of pairs of service s and its cost where P_{n-1} contains inputs of s and cost of s is its response time. P_n is a set of pairs of output parameters of all services in A_n and their costs united with the parameters in P_{n-1} where the cost of the parameter is the cost of its service plus the maximum cost between its service input parameters. if the parameter is already existed in P_{n-1} then recalculate its cost (steps from 4 to 11). Algorithm 4 assigns the current solution with the found solution if it has cost less than the previous one. Algorithm 5 assigns the SolutionList with the services that execute the currents solution. Algorithm 6 updates the graph by removing all parameters in P layer with cost greater than or equal to the current solution cost (parameters that surely lead to non-optimal solution). Algorithm 7 checks if the stop point is appeared or not to stop the whole algorithm. Stop point is appeared if the last P layer P_n contains the same parameters as in P_{n-1} layer. Algorithm 8 is responsible for printing the solution to the user on the screen in the order of calling.

For example, assume that services in Table 1 are all services in the system where $S = \{a, b, c\}$, and $T = \{d\}$. The solution *Sol1* in Figure 7 and the best parents listed in Table 2 are obtained using the proposed algorithm given in [9] while the solution *Sol2* in Figure 8 and the best parents listed in Table 3 are obtained using our proposed IoTMA algorithm. Compared both solutions, we found that both algorithms gave the same optimal solution that is $S \rightarrow \{S2\} \rightarrow \{S6\} \rightarrow T$.

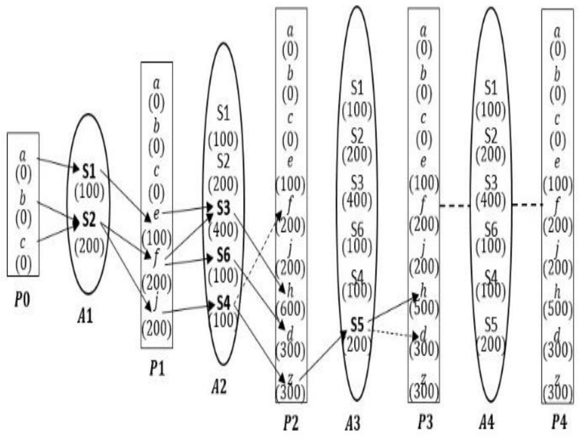
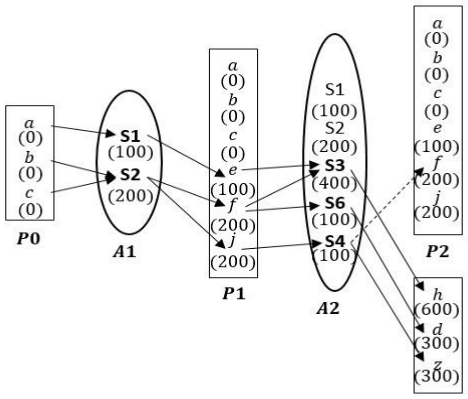


Figure 7. *Sol1* solution using the algorithm in [9]

Table 2. Best Parents List for $Sol1$

Parameter (p)	Parents	Best Parent	Cost
a	-	-	0
b	-	-	0
c	-	-	0
e	S1	S1	100
f	S2, S4	S2	200
j	S2	S2	200
h	S3, S5	S5	500
d	S6, S5	S6	300
z	S4	S4	300



a (0) b (0) c (0)
P0
S1 (100)
S2 (200)
A1
a (0) b (0) c (0) e (100) f (200) j (200)
P1
S1 (100) S2 (200) S3 (400) S6 (100) S4 (100)
A2
a (0) b (0) c (0) e (100) f (200) j (200)
P2
h (600) d (300) z (300)

Figure 8. *Sol2* solution using the proposed IoTMA Algorithm

Table 3. Best Parents List for $Sol2$

Parameter (p)	Parents	Best Parent	Cost
a	-	-	0
b	-	-	0
c	-	-	0
e	S1	S1	100
f	S2, S4	S2	200
j	S2	S2	200
h	S3	S3	600
d	S6	S6	300
z	S4	S4	300

From the two solutions, we can observe that our algorithm created a number of layers less than that presented in [9] and therefore it reduces the time needed to find the solution. This is because **UpdateGraph Algorithm** isolates all known non-optimal solutions paths early.

4. EXPIREMENTAL RESULTS

The mashup time needed to find the solution is affected by several factors such as size of the data set used in the system and the number and the size of layers created to find the solution, which is affected by:

- Number of input services S that $S\ parameters \sqsubseteq S.inputs$ ().
- The Solution Cost.

In our experiments, we create our data sets with different sizes due to the difficulty of having ready-made ones. Our data sets contains the service name, its input and output parameters, and its response time.

4.1 Data Set Size

We use four data sets for evaluation as: 40, 120, 200, 350 services with fixed solution cost = 800, and only one input services to conclude the effect of data set size on the algorithm time needed to find the optimal solution where *S* = {*aa*, *bb*} and *T* = {*D*}. The obtained results are listed in Table 4 where the time is measured in milliseconds.

Table 4. Algorithms time in case of different data set sizes and fixed number of layers (#layers)

DS	Size	Alg1 Time	IoTMA Time	Alg1 #layers	IoTMA #layers
1	40	5	5	5	3
2	120	5	5	5	3
3	200	6	6	5	3
4	350	6.5	6.5	5	3

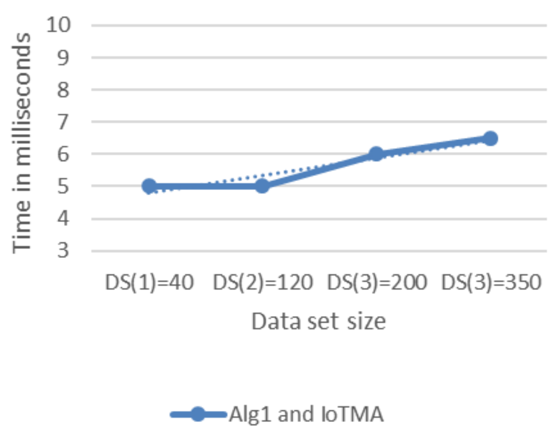


Figure 9. The effect of data set size on $Alg1$ and $IoTMA$

Figure 9 shows that the time needed by *Alg1* in [9] and *IoTMA* algorithm to find the optimal solution are equal and both are increased as the data set size is increased.

4.2 Number of Input Services

We conducted nine experiments with different number of input services, 200 Service as the used data set and fixed solution cost = 800, where S = {*a*, *b*} and T = {D} and the results are presented in Table 5.

Figure 10.a and Figure 10.b present the time needed by $Alg1$ and $IoTMA$ in case of 11 layers and 9 layers were created respectively. Both figures show that the time of both algorithms are approximately equal and is increased as the average layers size *(ALS)* is increased if the created layers number is fixed.

Table 5. Algorithms time in case of different number of input services, data set with 200 service

#Input Services	Alg1 Time	IoTMA Time	Alg1 #layers	IoTMA #layers	ALS
9	10.3	10.5	9	9	60
8	13.8	14	11	11	66
7	13.8	14	11	11	65
6	13.8	14	11	11	65
5	10.3	10.5	9	9	54
4	9.3	9.5	9	9	44
3	9.3	9.5	9	9	42
2	12.5	12.7	11	11	49

1	6	6	5	3	2

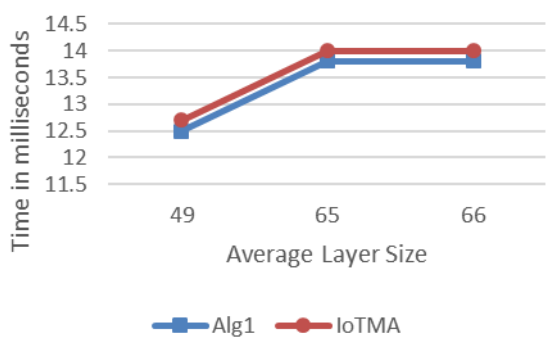


Figure 10.a. The effect of input services number on $Alg1$ and $IoTMA$ (the case of 11 layers)

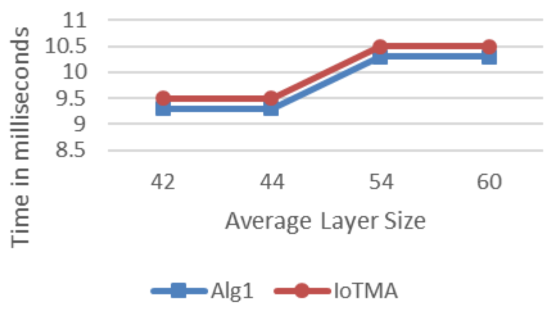


Figure 10.b. The effect of input services number on $Alg1$ and $IoTMA$ (the case of 9 layers)

4.3 Solution Cost

We test eight different solution costs using 9 input services with 200 services as data set where the maximum service *RT* is 800 and the minimum service *RT* is 100. If S = {*a*, *b*} and *T* = {*D*}, the results are presented in Table 6.

Figure 11 show that the solution cost has no any effects on *Alg1* algorithm time but has a strong effect on *IoTMA* time where it decreases as the solution cost decreases. This is because *IoTMA* algorithm creates number of layers less than *Alg1* due to the expected non-optimal solution paths reduction. Therefore, the solution with cost = 100 allows *IoTMA* to rejects large number of paths due to the large number of services has costs large than or equal to 100 (the desired cost), and this is not occurred when the solution cost equals 800.

Table 6. Algorithms time in case of different solution costs, input services = 9, Ds size = 200 service

Sol cost	Alg1 Time	IoTMA Time	Alg1 #layers	IoTMA #layers	Alg1 ALS	IoTMA ALS
800	10.3	10.5	9	9	60	60
700	10.3	8.6	9	7	60	46
600	10.3	8.6	9	7	60	46
500	10.3	8.6	9	7	60	46
400	10.3	8.6	9	7	60	46
300	10.3	7.7	9	7	60	34
200	10.3	6.3	9	5	60	14
100	10.3	6	9	3	60	8

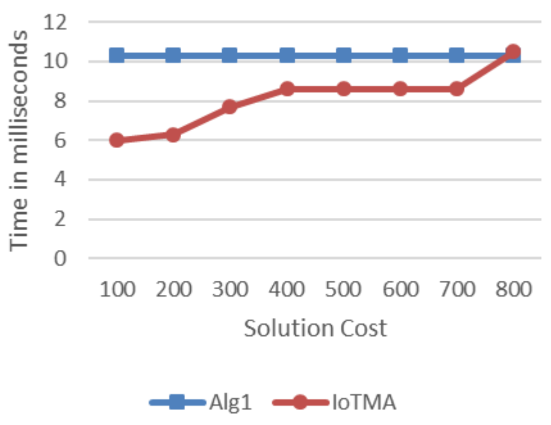


Figure 11. The effect of the solution cost on *Alg1* and *IoTMA*

5. CONCLUSON AND FUTURE WORKS

In this paper, we proposed an automatic *IoTMA* algorithm for mashup and *CoT* architecture in which mashup is provided as a service using *IoTMA*. By comparing the response time required by our algorithm and the other one presented in [9] to find the optimal solution, we observe that the two algorithms have approximately the same time even if the data set size and input services number are changed. However, *IoTMA* needs time less than the other one as the solution cost decreases. Accordingly, we conclude that in the best case, our *IoTMA* algorithm reduces the required response time by 42% of the other algorithm time, but in the worst case both algorithms need approximately the same response time to find the optimal solution. In the future, we will focus on implementing a cloud platform depending on our proposed CoT architecture and *IoTMA* algorithm that can provide mashup as a service.

6. REFERENCES

[1] A. Whitmore, A. Agarwal, and L. Da Xu (2014), "The Internet of Things - A survey of topics and trends." Information Systems Frontiers 17: 261-274.

[2] F. Belqasmi, J. Singh, S. Melhem, and R. H. Glitho (2012), "Soap-based vs. restful web services: A case study for multimedia conferencing." IEEE internet computing 16: 54-63.

[3] E. Pietroniro, and D. Fichter (2006), "Map mashups and the rise of amateur cartographers and mapmakers." ACMLA Bulletin127: 26-30.

[4] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu (2014). "Web services composition: A decade's overview." Elsevier Information Sciences 280: 218-238.

[5] D. Skogan, R. Groenmo, and I. Solheim (2004), "Web service composition in UML," Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004, pp. 47-57. USA-CA-Monterey.

[6] R. Gronmo, D. Skogan, I. Solheim, and J. Oldevik (2004). "Model-driven web services development," in Proceedings. IEEE International Conference on e-Technology, e-Commerce and e-Service, 2004, EEE-04, pp. 42-45. Taiwan-Taipei.

[7] S. Russell, and P. Norvig (1995), "Artificial Intelligence: A Modern Approach," Third Edition. Prentice-Hall, Inc.

[8] T. H. Cormen, C. E. Lieserson R. L. Rivest, and C. Stein, (2009), "Introduction to Algorithms," Third Edition, Massachusetts Institute of Technology.

[9] Y. Yan, and M. Chen (2015), "Anytime QoS-aware service composition over the GraphPlan", Service Oriented Computing and Applications, SOCA-Springer 9: 1-19.

[10] J. Zhou, T. Leppanen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin, and L. T. Yang (2013), "Cloudthings: A common architecture for integrating the internet of things with cloud computing," Seventeenth IEEE International Conference on Computer Supported Cooperative Work in Design, 2013. CSCWD 2013, pp. 651-657. USA-Canada-Whistler.

[11] S. Distefano, G. Merlino, and A. Puliafito (2013), "Towards the cloud of things sensing and actuation as a service, a key enabler for a new cloud paradigm," Eighth IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2013. 3PGCIC 2013, pp. 60-67. Italy-Capua

[12] S. Distefano, G. Merlino, and A. Puliafito (2012), "Enabling the Cloud of Things." Sixth IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2012. pp. 858-863. Italy-Palermo.

[13] J. Im, S. Kim, and D. Kim (2013), "IoT mashup as a service: cloud-based mashup service for the Internet of things," Tenth IEEE International Conference on Services Computing, 2013. SCC 2013, pp. 462-469. USA-Canada-Santa Clara

[14] T. Bhattasali, R. Chaki, and N. Chaki (2013), "Secure and trusted cloud of things," Annual IEEE India Conference, 2013. INDICON 2013, pp. 1-6. India-Kolkata.

[15] M. Blackstock, and R. Lea (2012), "IoT mashups with the WoTKit," Third IEEE International Conference on Internet of Things, 2012. IoT 2012, pp. 159-166. China-Jiangsu Province.

[16] R. Kleinfeld, S. Steglich, L. Radziwonowicz, and C. Doukas (2014), "glue.things: a Mashup Platform for wiring the Internet of Things with the Internet of Services," Fifth ACM International Workshop on Web of Things, 2014. pp. 16-21. USA-Cambridge.

[17] L. Atzori, A. Iera, and G. Morabito (2010), "The Internet of Things: A survey." Elsevier Computer Networks 54: 2787-2805.

[18] W.T. Tsai, X. Sun, and J. Balasooriya (2010), "Service-Oriented Cloud Computing Architecture," Seventh IEEE International Conference on Information Technology, 2010. pp. 684-689. USA-Las Vegas.

[19] A. Botta, W. de Donato, V. Persico, and A. Pescapé (2016). "Integration of Cloud computing and Internet of Things: A survey." Elsevier Future Generation Computer Systems 56: 684-700.

[20] Q. A. Liang, and S. Y. Su (2005), "AND/OR graph and search algorithm for discovering composite web services." International Journal of Web Services Research 2: 46-64.

[21] Y.-J. Lee, and J. S. Kim (2012), "Automatic web API composition for semantic data mashups," Fourth IEEE International Conference on Computational Intelligence and Communication Networks, 2012. CICN 2012, pp. 953-957. India-Uttar Pradesh.