A Distributed Visualization Service Composition System

Haihong E, Yimin Lin, Meina Song, Xiangyu Xu, and Chengcheng Zhang

***Abstract*—This paper designs a visual system and introduces an asynchronous mechanism based on message subscription, we can easily combine web services asynchronously or synchronously and publish them quickly. Besides, we also transform ordinary log content into event objects to simplify log analysis, integrate Swagger framework to maximize the ease of publishing composite services and deploy multiple system instances with Docker to solve the performance bottleneck problem in stand-alone deployment.**

***Index Terms*—Web service, service composition, visualization, asynchronously.**

I. Introduction

Web service is an application that interacts with a well-defined interface using standard application-layer web protocols and is based on a standard functional description language [1]. Web service technology is good at solving the integration problem between different platform applications well. From the perspective of service providers, the functions provided by the single web service are relatively simple, and often cannot meet the needs of developer directly [2]. A complete service usually needs to contain several basic web services. For example, a travel service may be composed of several services such as airline booking, hotel reservation, and car rental etc. [3]. Therefore, to meet the various requirements of developers, it is crucial to combine the scattered single web services to form a combined web service which can meeting a specified demand.

The service combination needs to integrate multiple independent and well-defined services into a whole service that satisfies the application requirements and functions with a certain mechanism [4]. Usually, Choreography or Orchestration aggregation strategy is adopted, and the CDL language (Choreography Description Language) and the BPEL language (Business Process Execution Language) are used to describe the interaction, cooperation or communication process between services [5], [6]. In fact, the above two languages belong to the underlying syntax-based description language [7]. As the number of interactive services increases, the complexity of the combined service will increase [7]. At the same time, the two languages require a central execution engine, such as ESB (Enterprise Service Bus), Etc. Hard to be used in a simple web service architecture [8]. This paper provides a flexible and easy-to-use GUI to support flexible combination based on existing web services, generating new combined web services. It's a fast and lightweight method of service combination in a web services architecture.

The main contributions of this article are:

1. Provide a flexible GUI to combine web services through drag-and-drop;

2. Introduce an asynchronous mechanism based on message subscription to combine web services asynchronously.

The organization of this paper is as follows:

In Section I, we introduce the framework of the overall system. In Section II, we introduce the implementation of drag-and-drop composite web service and data structure design. In Section III, we introduce the design and implementation of the execution engine and the asynchronous combination mechanism based on message subscription. In Section IV, we introduce the system test, and In Section III, we the summary of this article is given.

II. The Framework of Visual Web Service Composition System

By default, Consul is used as the service registry. All atomic web services (relatively simple web services) are retrieved from the service registry, and the combination of atomic web services is completed in a drag-and-drop manner. The framework of the system is as follows in Fig. 1.

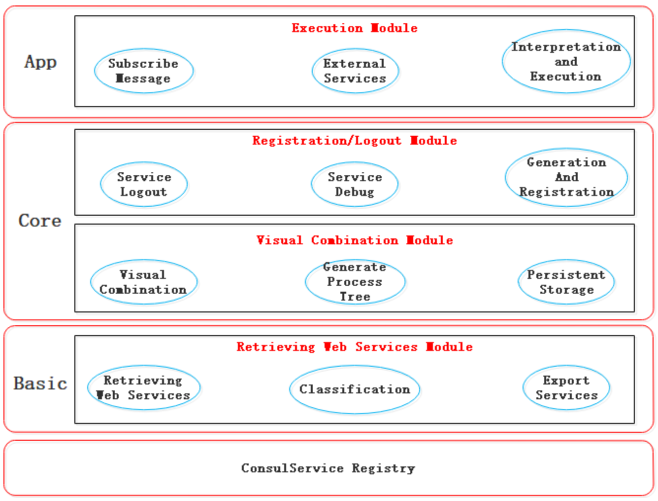


Fig. 1. System framework.

The processing time sequence of the system is divided into four parts: retrieving atomic Web service from service registry, visual composite web service, combined service debugging, registration and interpretation execution. The function flow chart is illustrated in Fig. 2:

1. Retrieving Atomic Web Service Module: Retrieving all atomic web services from the Consul Service Registry,

Manuscript received May 29, 2019; revised July 20, 2019.

Haihong E, Yimin Lin, Meina Song, Xiangyu Xu, and Chengcheng Zhang are with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: ehaihong@bupt.edu.cn, linyimin520812@gmail.com, mnsong@bupt.edu.cn, xuxiangyu_bupt@foxmail.com sweetzcc@163.com).

afterwards classifying them according to categories, and finally providing the services through the API interface;

2. Visual Combination Module: Retrieving the specified category of web service from the Atomic Web Service module, dragging to the visual composition GUI, and then clicking the web service button, selecting a specific atomic web service, next fill in the specific execution information of the atomic web service, and finally save the Atomic Web Service Information; continue with the above steps and combine them into a target composite web service process tree---process tree based on JSON.

3. Registration Module: Fill in the relevant information of the combined web service (combined service name, corresponding parameters, response value, path, etc.), and then save the relevant information to generate related composite services. Next debug the combined service. If the result of debugging is successful, invoke service registry API to register the combined service to the registration center. Otherwise, the relevant error information is returned and report the message to the administrator.

4. Execution Module: Developers invoke the composite web service, and then the parser parses the composite service process tree to retrieve atomic web service information and executes the atomic web service. If the atomic web service is synchronous, it’s executed directly and continues to parse the JSON process tree recursively. Otherwise, subscribes related message events, and then executes the asynchronous service once the related messages events are published, and next continues to parse the process tree based on JSON until the entire composition process is completed.

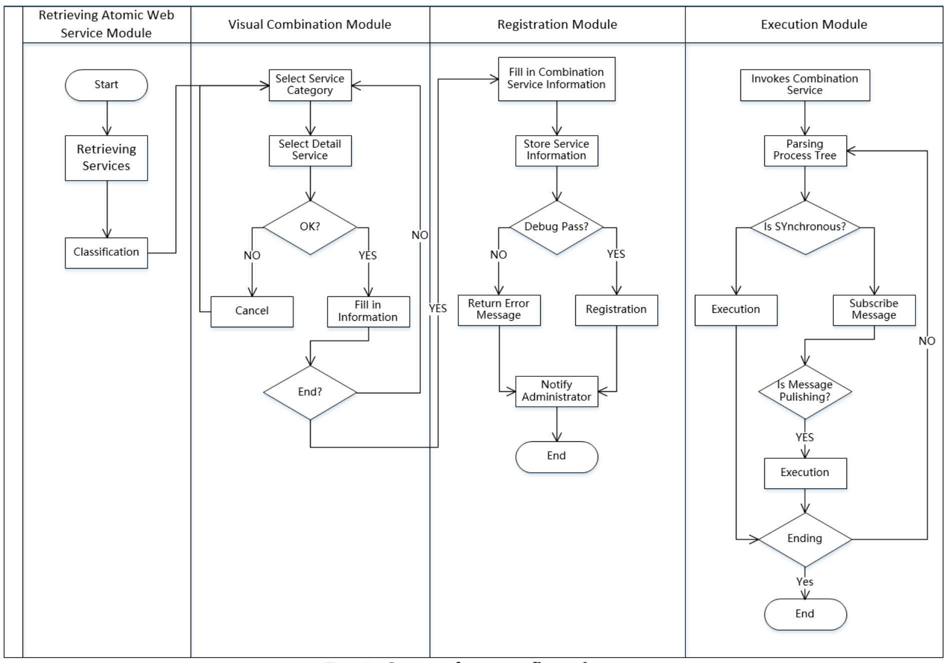


Fig. 2. System function flow chart.

III. Visualized Web Service Composition

A. Design of Visual Web Interface

We provide a visual web interface, and its functional structure is illustrated in Fig. 3. We can combine web services through dragging and dropping.

The specific functions are as follows:

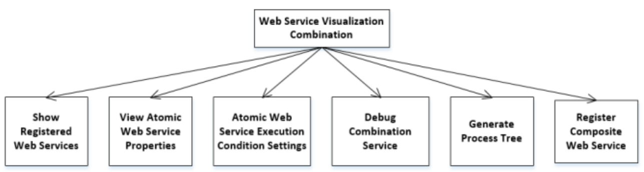
1. Viewing: The web services category in the left-hand menu bar read from the database dynamically and updates in real-time. After dragging the web service category icon to the combination area, click the icon to retrieve all atomic web services of the category. Attributes include: web service ID, description of returned value, name, type, event, parameter, status of online or offline, etc.

2. Setting: Set the execution conditions of the atomic web service (synchronous, asynchronous, and trigger conditions)

3. Debugging: When combination is completed, click “debug” menu, the debugging information is displayed in the console.

4. Generation: After successful debugging, click “Register” to generate a tree process file in JSON format. The process tree contains the execution flow of the composite service.

5. Registration: The generated tree process file interacts with the background, and the combined Web service is registered to the service registry.


Web Service Visualization Combination
Show Registered Web Services
View Atomic Web Service Properties
Atomic Web Service Execution Condition Settings
Debug Combination Service
Generate Process Tree
Register Composite Web Service
Fig. 3. Visual combination function.

B. Data Structure of Process Tree

In the process of composition, each atomic web service corresponds to a node in the process tree. We form an execution flow by specifying the parent-child relationship and conditions of execution between the atomic web services, and finally the execution flow stored as a process tree based on JSON, for the execution module to parse and execute.

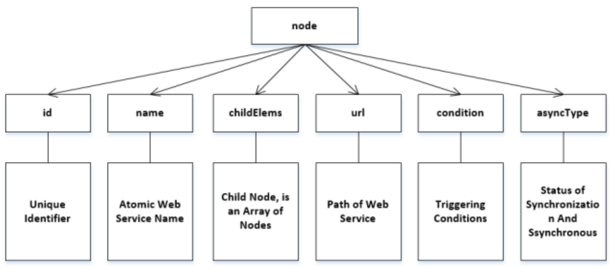


Fig. 4. Node properties.

The main attributes of node are shown in Fig. 4. The structure of the process tree based-on JSON is a recursive data structure. For example, the data structure corresponding to a combined service flow tree in Fig. 5 is as shown in Table I.

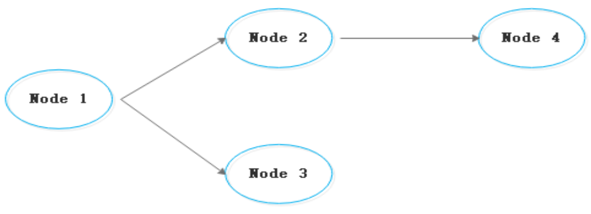


Fig. 5. Example of combination service.

TABLE I: DATA STRUCTURE OF PROCESS TREE

```
{
  "id": "001",
  "name": "node1",
  "url": "http://www.test.com/node1",
  "condition": "status=200",
  "asyncType": "0",
  "childElems": [
    {
      "id": "002",
      "name": "node2",
      "url": "http://www.test.com/node2",
      "condition": "status=200",
      "asyncType": "0",
      "childElems": [
        {
          "id": "004",
          "name": "node4",
          "url": "http://www.test.com/node4",
          "condition": "status=200",
          "asyncType": "0",
          "childElems": []
        }
      ]
    },
    {
      "id": "003",
      "name": "node3",
      "url": "http://www.test.com/node3",
      "condition": "status=200",
      "asyncType": "0",
      "childElems": []
    }
  ]
}
```

IV. EXECUTION ENGINE

The combined web service execution engine supports two execution mechanisms:

1. Synchronous interpretation and execution based on response status codes and response values.
2. Asynchronous interpretation and execution based on event and message publish/subscribe.

A. *Synchronous Interpretation and Execution*

The synchronous interpretation and execution based on the response status code and response value are based on the parent node execution result (response status code and value) to decide which child node should be executed, and then traverses in order until the node is a leaf node which means completing the execution of the entire process. As shown in Fig. 6, if condition 1 is triggered, Service 2 is executed. The response of Service 2 is the condition required for Service 5 execution. Therefore, Service 5 is executed after Service 2 finished. And then the response of Service 5 is the required condition for Service. 7 Executing, so then Service 7 will be invoked, and finally the entire process ends.

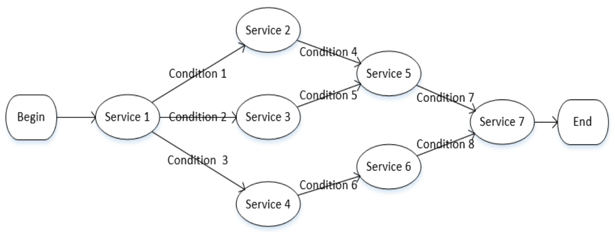


Fig. 6. Synchronous interpretation and execution.

B. *Asynchronous Interpretation and Execution*

The asynchronous interpretation and execution based on event publishing/subscription is divided into two processes: publication of messages and subscription of messages. Publication of messages is completed by a specified interface. When a message needs to be published, the interface can be invoked to complete the publication. The subscription of the messages is completed by the interpretation executor. When traversing the asynchronous node (when the asyncType is equal to 1, a property of the node), the related message will be subscribed. This paper uses RabbitMQ message middleware to achieve message subscription and release as shown in Fig. 7. RabbitMQ is a mature message queue middleware which implements AMQP (Advanced Message Queuing Protocol).

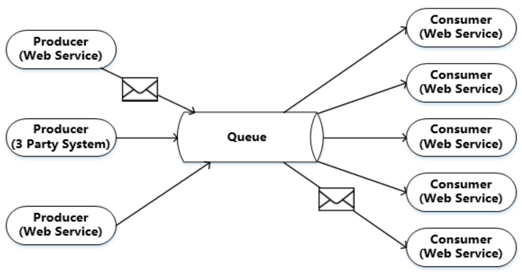


Fig. 7. Message publish and subscribe.

C. *Implementation of the Execution Engine*

The execution engine is a hierarchical traversal operation on the JSON process tree actually. The difference from the hierarchical traversal is that the recursive operation of the executor is a conditional traversal, and the corresponding operations are performed according to the relevant conditions. The execution process is as follows:

1. Invoking the root node: access the atomic web service corresponding to the root node, and obtain the response.
2. Traversing the child node of the node

a) judging whether the child node is a synchronous node or an asynchronous node according to the asyncType field
   i. If it is synchronous, according to the response status code and response of the parent node, compares with the condition specified in the child node, invokes the node which condition is satisfied.
   ii. If it is asynchronous, subscribes to the message specified in the child node

3. Repeat step 2 until the traversal of process tree is completed.
4. Related systems publish message events
5. The node which subscribes to the relevant message is invoked.
6. Repeat step 2 until the traversal of the process tree is completed.

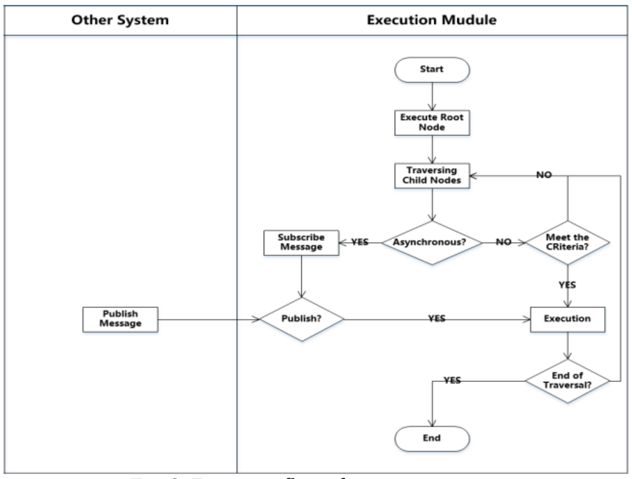


Fig. 8. Execution flow of execution engine.

The pseudo code of execution engine is shown in Table II.

TABLE II: PSEUDO CODE OF EXECUTION ENGINE

```
function run(node) {
   //  Invoke API on the node
   call(currentUrl)
   // Traversing child nodes
   let children = node.children;
   // If child node exists
   if(children) {
      for(let i = 0; i < children.length; i++){
            if(API is synchronous){
                  // Scribes message
                  subscribe(event);
                    // Continue to traverse the child node
                 run(children[i]);
               }else{
                  if(response is in need){
                // Continue to traverse the child node
            run(children[i]);
                  }
         }
   }
}
```

V. LOG, DOCUMENT AND DEPLOYMENT

A. Logs and Error Handling

When the combined service is online, it is necessary to control and supervise the service and its related atomic services to ensure the stable operation of the service.

The records accessed by the user are recorded in the log files asynchronously, and retrieving data through the log analysis framework, then perform statistical analysis and visualization. Log analysis includes the overall access to the composite service and all atomic services.

Traditionally, we are directly outputting the level of the log and the content string of the log(Message). However, we not only pay attention to when the composite service is called, what the result is, but also the access details and context of the related atomic services, as well as the associated log. As shown in Fig. 9. If the call fails, it is determined according to the response result whether the combined service needs to be offline. Therefore, it is necessary to carry out transformation on the traditional log structure. The method of transformation is to extract the key fields of the log into an event object directly [9]. The data structure of the object is as follows:

```
{
   "datetime":string,
   "level":string,
"type":string,
 "reqId": string,
   "reqUid": string,
   "data":{
      "url":string,
      "ip":string,
      "method":string,
      "userAgent": string,
      "headers": string
   },
   "browser": string,
   "os":string,
   "server":string
}
```

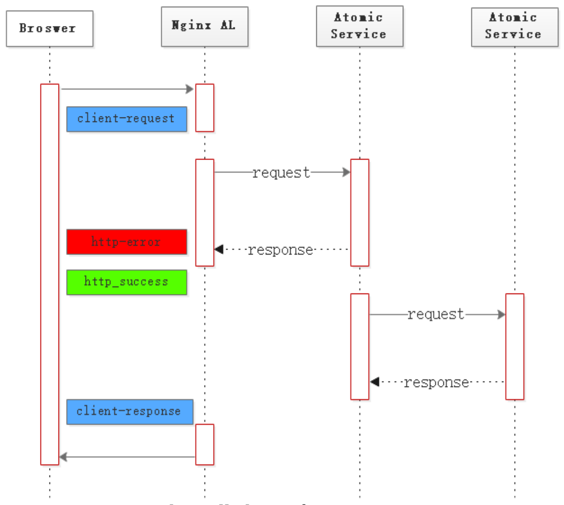


Fig. 9. The Call chain of composite service.

Most of the time, the log will be output to the local disk, and then directly use Linux commands to troubleshoot the problem, but the combined system supports distributed deployment, will have multiple machines. There are some troubles to locate and troubleshoot problems directly, so it is necessary to have a platform management log. The method we use is to collect all logs into Kafka, and then import them into Elasticsearch and HDFS respectively, and do real-time search, analysis, offline statistics and data backup. The following describes the use of log analysis to complete the offline of invalid combination services. The flow is shown in Fig. 10.

First, when the composite service invoked, will generate a unique identification request ID, and generate a request log, and then the relevant atomic services are invoked. If some atomic services go offline or some errors occurs, the system will look for the same service deployed on other servers and call it again. If there is no alternative service, generate an error log of the atomic service and the combined service, then the combined service is offline and removed from the registry, and finally the error message and offline message are pushed to the administrator. If the invoke is successful, the response log is logged.

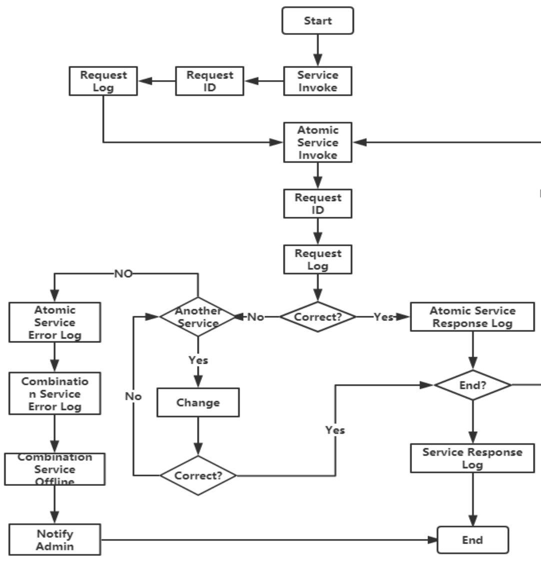


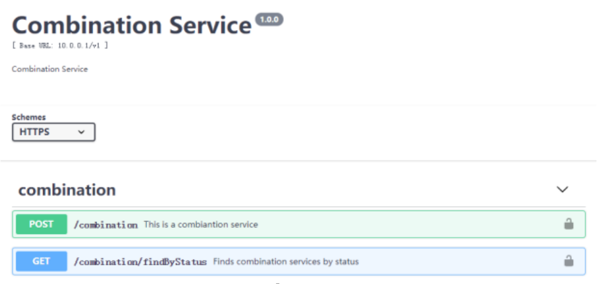
Fig. 10. The flow of error handling.

B. *Specification Swagger Document Framework*

Swagger is a canonical and complete framework for generating, describing, invoking, and visualizing RESTful Web services [10].

This paper integrates Swagger framework into our composite system with the goal of maximizing the ease of publishing and use of composite services. When the combined system is running, it will subscribe to registration, logout and offline events. When registering or unregistering services, it will release related events, and then trigger system to modify the Swagger API specification document, and finally Generate an APIs document based on JSON and provide real-time preview. Developers can use the Swagger framework to view and test the composite services. All of this is done automatically, without any intervention from the system user. As shown in Fig. 11, a real-time online preview of the composite services using Swagger UI. And the logic of real-time update of Swagger document is shown in Fig. 12.

At the same time, the paper also integrates a code generator to generate software development kits (SDK) for various languages (including Java, Objective-C, PHP, Python, etc.) according to the Swagger specification document. The generated SDK encapsulates the HTTP invoke to the REST API service defined in the Swagger specification document. So developers can use the language they are familiar with to invoke the function or class method to complete the service call without having to deal with the detail of underlying HTTP transport.


Combination Service 1.0.0

[Base URL: 10.0.0.1/v1]

Combination Service

Schemes: HTTPS

combination

POST /combination This is a combiantion service

GET /combination/findByStatus Finds combination services by status

Fig. 11. The swagger UI.

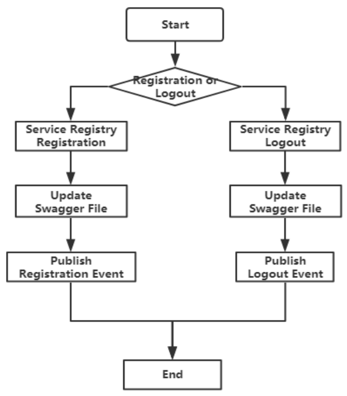


Fig. 12. The logic of real-time update of swagger document.

C. *Distributed Deployment Base on Docker*

Because of the performance bottleneck problem in stand-alone deployment, this paper introduces Docker and Nginx load balancing technology, encapsulates the entire combined system into a Docker image, shields the environment differences of different servers, and implements rapid deployment of system multi-instance using Docker container [11].

Docker uses Google's Go language for development and implementation. Based on the Linux kernel's cgroup, namespace and greatly simplifying the creation and maintenance of containers. Docker technology is lighter and faster than virtual machine technology.

The Docker file that encapsulates the system into a Docker image is shown in Table:

```
FROM node:8.11.1
ADD . /root/api-gateway-combination/
WORKDIR /root/api-gateway-combination
RUN npm install -g ts-node typescript && npm install
EXPOSE 8001
CMD ["ts-node", "./src/app.ts"]
```

The first line uses node:8.11.1 as the base image, the second and third lines add the source code to the image, the fourth and fifth lines install the typescript environment, the sixth line exposes the port, and the seventh line runs the main program.

Multi-instance deployment may have the consistency problem of memory data. This paper mainly uses the event subscription/release mechanism to ensure data consistency. As shown in Fig. 13, when a new instance is run, the registration, logout, and other message events of the composite service are subscribed. When there is a new combined registration or a combined service is logged out, all instances update the relevant memory data and all data is consistency in all instances.

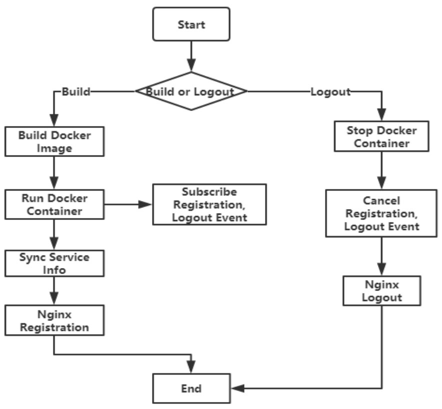


Fig. 13. Distributed deployment base on docker.

VI. System Testing

We have conducted a set of experiments that aim to evaluate the performance of our architecture during the execution of atomic web service and the combined service. In these experiments, all web services were executed by our architecture in centralized configurations on Alibaba Cloud ECS. Fig. 14 displays a set of graphs that provide the service execution time, total transaction rate and the number of concurrency for each experiment. The performance analysis verifies our research hypothesis, and shows that our approach is highly efficient, and scales accordingly with the increasing size of ECSs. And the visual web service composition page is shown in Fig. 15.

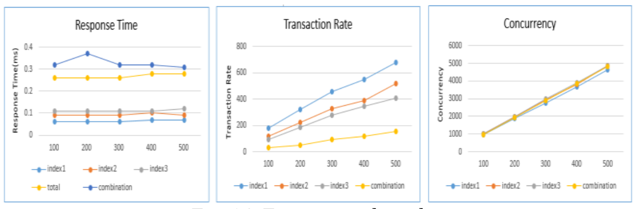
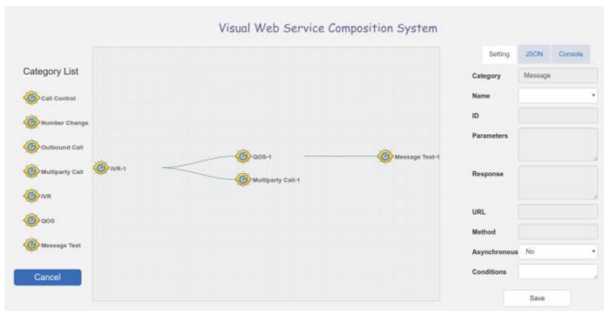


Fig. 14. Experimental results.



Category: Message
Name:
ID:
Parameters:
Response:
URL:
Method:
Asynchronous: No
Conditions:

Fig. 15. Visual web service composition.

VII. Conclusion

In this paper, we propose a system based on a novel simplified web combination method. The system is based on the B/S architecture mode, provides a flexible visual interface, implements a drag-and-drop composite Web service based on JSON tree data structure, and introduces an asynchronous mechanism based on message subscription to realize asynchronous combination of Web services.

We note that a shorter conference version of this paper appeared in ICIM (2019). Our initial conference paper did not address the problems of log analysis, document sync and distributed deployment. This manuscript addresses these issues and provides additional analysis on the error handling using log analysis to improve the system stability.

Acknowledgment

This work is supported by the Ministry of Education-China Mobile Science Research Foundation (MCM20170210); Engineering Research Center of Information Networks, Ministry of Education.

References

[1] Y. Xin, J. Li, and Z. Li, "Research progress of microservices composition methods," *Wireless Communication Technology*, vol. 27, no. 3, pp. 42-46, 2018.
[2] E. B. H. Yahia, L. Réveillère, Y. D. Bromberg *et al*., "Medley: An event-driven lightweight platform for service composition," in *Proc. International Conference on Web Engineering*, 2016, pp. 3-20.
[3] Y. Zhang, T. Lei, H. Gao, and Y. Ding, "Instant service recommendation research in exploratory service composition environment," *Journal of Chinese Computer Systems*, no. 5, 2017.
[4] J. Wang and Y. Zhang, *Utilizing Marginal Net Utility for Recommendation in E-commerce*, 2011.
[5] J. G. Dai, "Many-server queues with customer abandonment: A survey of diffusion and fluid approximations," *Journal of Systems Science and Systems Engineering*, no. 1, 2012.
[6] B. Li, D. Qiu, H. Leung, and D.Wang, "Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph," *Journal of Systems & Software*, no. 6, 2012.
[7] X. Xu, Z. Wang, D. Guo, Y. Wang, and Y. Kang, "Design and implementation of a dynamic data service publishing engine," *Computer Applications and Software*, vol. 35, no. 7, pp. 126-130, 177, 2018.
[8] A. Brogi, S. Corfini, and R. Popescu, "Semantics-based composition-oriented discovery of web services," ACM Transactions on Internet Technology, no. 4, 2008.
[9] Summary of node framework access ELK practice. [Online]. Available: https://zhuanlan.zhihu.com/p/57340438
[10] Model APIs with accuracy. [Online]. Available: https://swagger.io/solutions/
[11] What is docker. [Online]. Available: https://opensource.com/resources/what-docker

E Hai Hong was born in 1982 and received her Ph.D. degree from Department of Computer Science, Beijing University of Post and Telecommunication, in 2010.

She is now serves as an associate professor in Department of Computer Science, Beijing University of Post and Telecommunication, Beijing. Her research interests include service computing and service engineering, big data and AI.

Dr. E Publishes more than 60 SCI/EI academic papers, applied for 43 invention patents, wrote more than 20 national and industry standards.

Lin Yi Min is a master student in Beijing University of Post and Telecommunication. His research interests include big data, data service, distributed system and web development.

Song Mei-Na was born in 1974, Ph.D., professor. Her current research interests include service computing and service engineering, big data and AI.

Xu Xiangyu was born in 1995, Panjin, Liaoning Province. She was admitted to Beijing University of Posts and Telecommunications in China in 2013. Now she is doing her master degree in computer science and technology at BUPT. Her research interest is the big data visualization.

Zhang ChengCheng was born in 1993, and now is a master student in Beijing University of Post and Telecommunication. His research interests include container management, micro-service architecture and system architecture.