# A METHOD FOR EFFICIENT EXECUTION OF BIOINFORMATICS WORKFLOWS

JUNYA SEO[1] YOSHIYUKI KIDO[2] SHIGETO SENO[1]
j-seo@ist.osaka-u.ac.jp y-kido@ist.osaka-u.ac.jp senoo@ist.osaka-u.ac.jp

YOICHI TAKENAKA[1] HIDEO MATSUDA[1]
takenaka@ist.osaka-u.ac.jp matsuda@ist.osaka-u.ac.jp

[1] *Department of Bioinformatic Engineering, Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan*
[2] *The Center for Advanced Medical Engineering and Informatics, Osaka University, 2-2 Yamadaoka, Suita, Osaka 565-0871, Japan*

Efficient execution of data-intensive workflows has been playing an important role in bioinformatics as the amount of data has been rapidly increasing. The execution of such workflows must take into account the volume and pattern of communication. When orchestrating data-centric workflows, a centralized workflow engine can become a bottleneck to performance. To cope with the bottleneck, a hybrid approach with choreography for data management of workflows is proposed. However, when a workflow includes many repetitive operations, the approach might not gain good performance because of the overheads of its additional mechanism. This paper presents and evaluates an improvement of the hybrid approach for managing a large amount of data. The performance of the proposed method is demonstrated by measuring execution times of example workflows.

*Keywords*: web service, workflow, data transfer, database.

## 1. Introduction

Efficiently executing large-scale and data-intensive workflows common to scientific applications must take into account the volume and pattern of communication. Traditionally, two common architectures for implementing workflow are proposed. Those are centralized orchestration and choreography [3].

Figure 1 shows the two traditional architectures. Centralized orchestration describes how services can interact at the message level, with an explicit definition of the control and data flows. This architecture has a workflow engine to execute workflows. The engine acts as a controller of the involved services. Both control and data flow messages are passed through this central engine. Examples of such engines in bioinformatics domain are Taverna [6] and Triana [8].

On the other hand, a choreography model describes a peer-to-peer collaboration between collections of services. This architecture basically does not have a centralized engine that controls workflows. Choreography focuses on message exchange, all involved services are aware of their partners and when to invoke operations. The

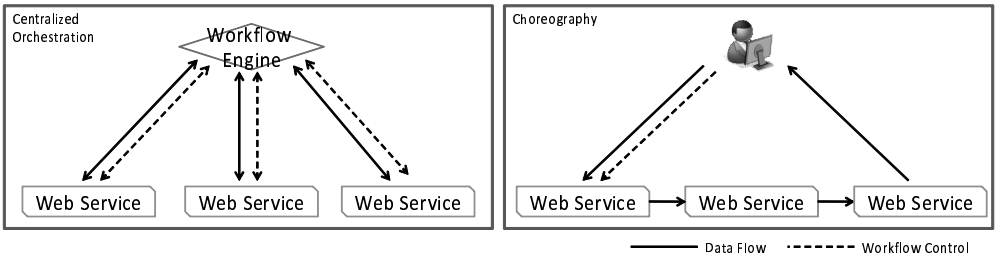


Fig. 1. Traditional architectures for workflow execution.

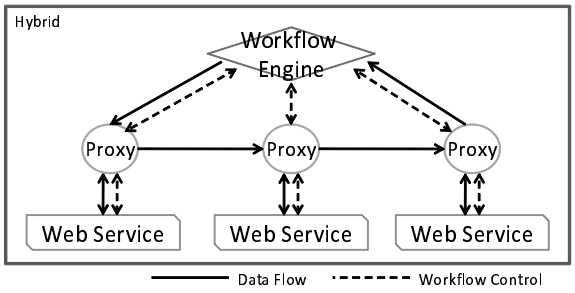


Fig. 2. Hybrid architecture of centralized orchestration and choreography.

Web Services Choreography Description Language (WS-CDL) [4] is an XML-based language for describing this model.

Centralized control is suitable for scientific workflows because those workflows contain many services with complicated controls (such as conditional branches). However, in a centralized engine, every resulting output of each service is sent back to the engine. Thus the engine can easily become a performance bottleneck by the concentration of returning data. While a choreography architecture can optimize data transport between services, but the implementation of this architecture is highly complex and rigid.

Baker *et al.* proposes a hybrid architecture that combines an orchestration with a centralized controller (a workflow engine) and a choreography model of distributed data transport [2]. Figure 2 shows an image of the architecture. The architecture mixes benefits of the above two architectures. It introduces proxies between Web services and a workflow engine. The engine sends a request to a proxy instead of a service. The proxy invokes the service and save its result to a local storage, and returns a unique ID of the result to the engine. The result (intermediate data) is not sent back to the engine but is transferred directly to other proxies according to the data flow in the workflow. Therefore the engine does not have to receive the intermediate data to be sent to the next service.

In this paper, we propose a modification of the hybrid architecture. The hybrid

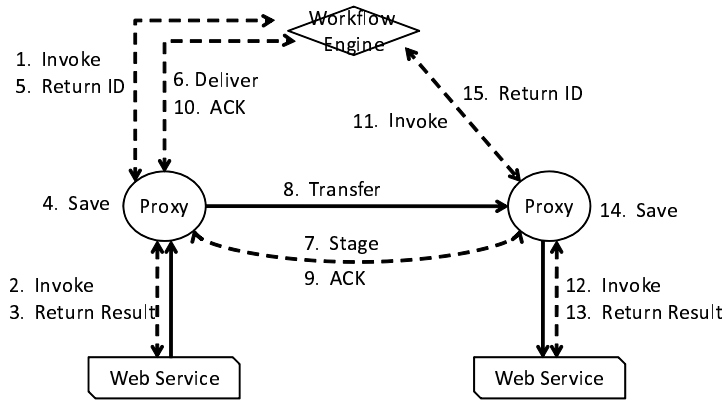


Fig. 3. Execution steps of a workflow with two Web services in the original hybrid architecture.

architecture can suppress the amount of data transfer by removal of the transfer to/from the engine. Thus its performance is highly improved when the amount of the resulting output of each service is very large. However, some overheads exist in the hybrid architecture at the proxies. When it is adopted to a workflow including repetitive operations (such as loop operations), the engine need to frequently interact with proxies for invoking services and transferring their results to other proxies. We change the protocol between the engine and proxies for reducing the overheads as described in Sec. 3.

## 2. Workflow Operations in Hybrid Architecture

Workflows are frequently used in various scientific fields. Especially, in the bioinformatics domain, workflows can be used for integrating a number of remote resources in terms of data and applications [1]. Many analysis tools are available as Web services from EBI [5, 9], DDBJ [10], and KEGG [11]. Scientists often use several databases for their analyses and compose workflows with several Web services that are distributed geographically.

Figure 3 shows the execution steps in a workflow containing two Web services in the hybrid architecture [2]. Those steps are described as follows in detail.

1: A workflow engine invokes a proxy to make an invocation to a Web service (Fig. 3 left).

2: The proxy invokes the Web service.

3-4: The proxy receives the result of the service, and save it within the proxy. There is a requirement that the proxy has enough disk space to store the result.

5: The engine receives only the ID of the result from the proxy instead of its whole data.

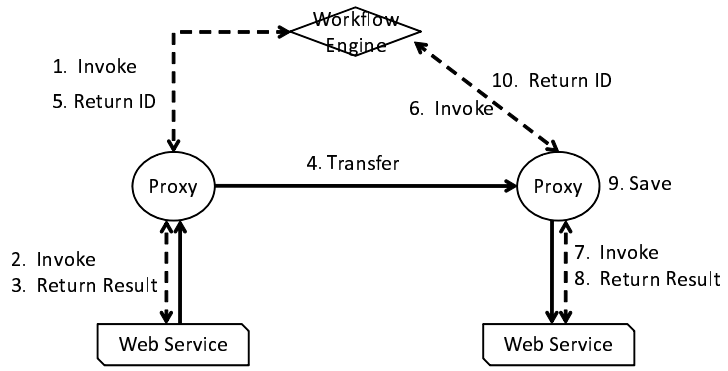


Fig. 4. Modified execution steps of a workflow in the proposed hybrid architecture.

6: The engine invokes the proxy again to make transfer the result (the operation is named *deliver* in Fig. 3).

7: The proxy invokes another proxy (Fig. 3 right) to notify data transfer.

8: The second proxy receives the result from the first proxy (the operation is named *stage* in Fig. 3).

9-10: After the data transfer finished, the second proxy returns acknowledgment to the first proxy, and the first one also returns it to the engine.

11-15: The engine invokes the second proxy with the ID, and the second proxy invokes another Web service.

As shown in Fig. 3, additional invocation steps are necessary compared with a centralized orchestration (2, 6, 7, and 12). If the workflow includes repetitive operations, those extra invocations are carried out at the number of repeats. The extra invocations may increase the processing time of the workflow.

## 3. Improved Method

As the amount of biological databases has been increasing rapidly, the data to be managed in workflows has also been increasing. Those databases contain huge numbers of entries. For example, there are over 9 million entries are registered in the UniprotKB database [12]. When scientists want to analyze such a huge number of data in their workflows with Web services, repetitive invocations of the services are occurred. If they use several databases in combination, the total number of resulting data should be very large. Those repetitive invocations in a workflow cause large overheads in the hybrid architecture.

To decrease the number of the extra operations described in the previous section, we present a modification to change the execution steps in the hybrid architecture. The modified execution steps are described as the followings (see Fig. 4).

1: A workflow engine invokes a proxy to make an invocation to a Web service (Fig. 4 left).

2: The proxy invokes the Web service.

3-4: The proxy receives the result of the service, and *transfers it to another proxy* (Fig. 4 right).

5: The engine receives only the ID of the result from the first proxy instead of its whole data.

6-11: The engine invokes the second proxy with the ID, and the second proxy invokes its Web service. If it is not necessary to transfer the result of the Web service to other proxies, the proxy saves the result within the proxy.

In the proposed modification of the execution steps, we decrease overhead at the following two points:

- Transfer the result of a Web service to the next proxy directly.
  A proxy transfers the result to the next proxy directly. In this case, the proxy does not need to save the result within the proxy. Thus one step is removed from previous hybrid architecture (Step 4 in Fig. 3).
- Remove deliver and stage operations.
  By transferring the result directly, the workflow engine and the first proxy does not need to invoke the first proxy and the second proxy, respectively, to make transfer the result. Thus deliver and stage operations and receiving their acknowledgments (Steps 6, 7, 9 and 10 in Fig. 3) can be removed.

Totally we can remove two Web service invocations and one data saving within the first proxy. The proxy we have proposed is deployed as a Web service as the same as the original hybrid architecture. Figure 5 shows the deployed operation in our proxy. Our proxy has only one operation to invoke a Web service and to make data transfer.

The operation "invoke" receives 4 arguments. "wsdl", "operation" and "args" are the same as those in the original hybrid architecture. There is a difference at argument "returns". Our operation can handle multiple results so that the workflow engine can extract resulting data to be returned. The destination information of the data transfer is also included in "returns" like:

```
resultA->hybrid.ist.osaka-u.ac.jp
resultB->http://hybrid.ist.osaka-u.ac.jp:8080/axis2/services/Hybrid?wsdl
```

Users can specify transferring destination as its host name or WSDL location. Users also can specify different destinations in each result.

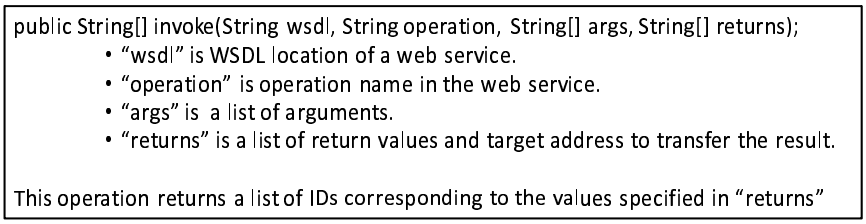


Fig. 5. Deployed operation in the modified proxy.

## 4. Experimental Result

### 4.1. *Experiment with Test Web Services*

In order to evaluate the proposed method, we have carried out performance analysis tests. In this analysis test, we prepared two test Web services.

- WS1 (Data Generate Service): This service receives an integer as its parameter, and generates a random string whose length is the parameter and returns them.
- WS2 (Echo Size Service): This service receives a string and returns its size.

We set two proxies and one workflow engine. The two proxies are located in the same local area network to the two Web services, respectively. The engine is set at a distant place from those proxies so that the engine communicates with the proxies through a wide area network. We composed a simple workflow that generates a string and counts the size. First, the engine calls WS1 and gets a string. After that, the engine passes the string to WS2.

We compare our method with a centralized orchestration and the original hybrid architecture changing the data size and the number of data.

Figure 6 shows the result of small data in the experiment. The data size varies from 1k to 1000k bytes per 100k bytes. In this experiment, the workflow handles only one single data.

In both the original and the proposed hybrid architectures, there is an initial overhead when the workflow engine invokes a proxy instead of a Web service, but the data transfer time sending its result back to the engine can be reduced since the result can send it to another Web service via their proxies without sending through the engine. It is to be noted that we assume a Web service and its proxy are located within the same local area network and the data transfer speed between them is fast.

If the size of the resulting data is larger, the data transfer overhead via the engine is getting larger in the centralized orchestration architecture. Thus the execution time of the centralized one is longer than those of the two types of the hybrid architecture when transferring data size is more than 400k bytes as shown in Fig. 6.

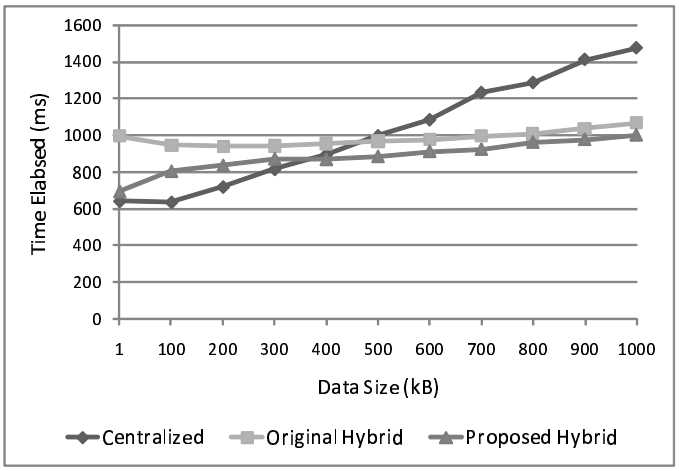


Fig. 6. Execution times on small data.

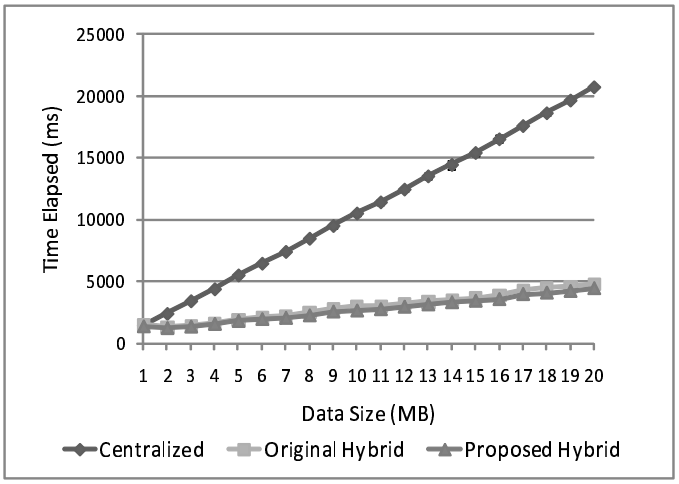


Fig. 7. Execution times on large data.

Figure 7 shows larger data pattern in the experiment. From the result, both the original and proposed hybrid architectures achieve higher performance compared with the centralized orchestration. This indicates that the hybrid architecture is suited for workflows having larger data communication between Web services. In both of small and large data patterns, Figs. 6 and 7 show that the proposed architecture takes slightly shorter execution time compared with that in the original hybrid architecture.

Figure 8 shows performance result on the number of data. We measured it for five types of the data of which numbers ranging from 1 to 500. In all the cases, the

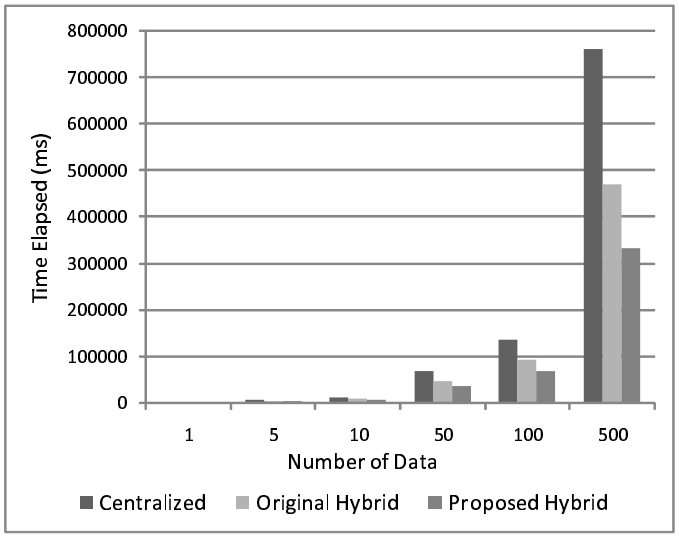


Fig. 8. Execution times on the number of data.

data size is set to 1M bytes. Figure 8 shows that, the proposed hybrid architecture achieves higher-performance compared with the other ones when the number of data is getting larger due to the reduction of the execution steps in the proposed architecture.

The performance at a large number of data is important since there are often many repetitive operations in bioinformatics workflows when they process a large number of data (such as, data for genes and proteins). The proposed architecture is suitable for the situation managing such many repetitive operations.

On the other hand, the proposed architecture contains the same overhead in invoking proxies as the original hybrid architecture. If the data for each operation is small (less than 400k bytes) in its size, the centralized orchestration architecture gains better performance.

### 4.2. *Experiment with Bioinformatics Web Services in Distributed Environment*

We measured performance for executing a workflow composed of bioinformatics Web services. The Web services we used are getEntry and cpgreport. The getEntry service receives an entry ID of the DDBJ sequence database [10] and returns the content of the entry identified with the ID. Whereas, the cpgreport service receives a nucleotide sequence data and returns the locations of CpG-rich regions in the sequence. The service was built by using the EBI Soaplab Web Services [7] for the EMBOSS programs [13].

For this experiment, we used the following two sets of DDBJ entries. For extracting the above DDBJ entries associated with the GO terms, we used the ARSA

Table 1. Execution times of bioinformatics Web services

| Data set | Data size | Centralized [a] | Original hybrid [a] | Proposed hybrid [a] |
|---|---|---|---|---|
| Set 1 | 190MB | 410 | 308 | 250 |
| Set 2 | 293MB | 703 | 638 | 532 |

*Note*: [a] Measured in seconds.

service at the DDBJ Web services. The execution time of the ARSA service is not included in the experiment.

- Set 1: 190 DDBJ entries associated with Gene Ontology GO:0007010 (cytoskeleton organization).
- Set 2: 466 DDBJ entries associated with Gene Ontology GO:0016747 (transferase activity, transferring acyl groups other than amino-acyl groups).

The workflow invoking the two Web services is carried out with many repetitive operations. Both services are invoked at the number of the DDBJ entries of the above sets. The data size of each entry ranges from 1k to 5M bytes.

We placed a workflow engine at the National Institute of Informatics, Tokyo, and two Web services at the two campuses (Suita and Toyonaka) of Osaka University, Osaka. In both the original and the proposed hybrid architectures, proxies are placed one by one at the same local area network of the Web services. The average round trip times are 10.1 ms (milliseconds) between Tokyo and Suita, and 1.1 ms between Suita and Toyonaka. The average file-transfer speeds measured with FTP are 37 Mbps between Tokyo and Suita, 94 Mbps between Suita and Toyonaka. The engine and Web services are composed of Linux workstations with Intel Xeon or Core 2 Duo CPUs (2.0GHz or 2.53GHz clock). The Web services were developed in Java, with the Apache/Tomcat software.

Table 1 shows the execution times of the workflow in the distributed environment. "Data size" in Table 1 means the total amount of the output of the getEntry service. According to Table 1, the proposed hybrid architecture gains better performance than those of the other architectures when a bioinformatics workflow to be executed has many repetitive operations.

## 5. Discussion

We have proposed a modification of the hybrid architecture for workflow execution, which is suitable for bioinformatics workflows including repetitive invocations. When the number of data is increased, the advantage of the proposed architecture is also increased.

The data size is important to select the architecture for workflow execution. The experiment results show that the centralized orchestration has advantage if the amount of data to be transferred in a workflow is relatively small. If a workflow need to transfer large data between distantly-located Web services to be invoked at

a number of times, our architecture gains better performance.

In this paper, we evaluated only simple repeat patterns of workflows. More types of scientific workflows including conditional branches should be evaluated as future works. For example, when a workflow includes a conditional branch and it need to select the destination of the data transfer by analyzing the result of a Web service, the proxy of the Web service cannot simply decide the destination of the transfer. We will consider the further improvement of the architecture for resolving this issue in the future.

## References

[1] Addis, M., Ferris, J., Greenwood, M., Li, P., Marvin, D., Oinn T., and Wipat, A., Experiences with e-Science workflow specification and enactment in bioinformatics, *Proc. UK e-Science All Hands Meeting*, 459–466, 2003.

[2] Baker, A., Weissman, J., and Hemert, J., Eliminating the middleman: peer-to-peer dataflow, *Proc. 17th Intl Symp. High Performance Distributed Computing*, 55–64, 2008.

[3] de Knikker, R., Guo, Y., Jin-long L., Kwan, A. K. H., Yip, K. Y., Cheung, D. W., and Cheung, K.-H., A web services choreography scenario for interoperating bioinformatics applications, *BMC Bioinformatics*, 5(25), 2004.

[4] Kavantzas, N., *et al.*, *Web services Choreography Description Language (WS-CDL) Version 1.0*, 2005.

[5] Lbabarga, A., Valentin, F., Anderson, M., and Lopez, R., Web services at the European Bioinformatics Institute, *Nucleic Acids Research*, 35:6–11, 2007.

[6] Oinn, T., *et al.*, Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, 20:3045–3054, 2004.

[7] Senger M., Rice P., Bleasby A., and Uludag M., Soaplab: open source web services framework for bioinformatics programs, *Proc. 10th Annual Bioinformatics Open Source Conf.*, 2009.

[8] Taylor, I., Shields, M., Wang, I., and Philp, R., Distributed P2P computing within Triana: a galaxy visualization test case, *Proc. IPDPS 2003*, 16–27, 2003.

[9] `http://www.ebi.ac.uk/Tools/webservices/`

[10] `http://www.xml.nig.ac.jp/index.html`

[11] `http://www.genome.jp/kegg/soap/`

[12] `http://www.uniprot.org/help/uniprotkb`

[13] `http://emboss.sourceforge.net/`