# Big-Data Approaches for Bioinformatics Workflows: A Comparative Assessment

Rickey T.P. Nunes[1](✉) and Santosh L. Deshpande[2]

[1] VTU Research Resource Centre, Visvesvaraya Technological University, Belagavi, India
rickeynunes@gmail.com

[2] Centre for Postgraduate Studies, Visvesvaraya Technological University, Belagavi, India
sld@vtu.ac.in

**Abstract.** There is a big-data explosion in the field of bioinformatics, with the rapid growth in the size of biological data. In bioinformatics, workflows are used to integrate and analyze biological data. Orchestration and choreography are the two approaches used to execute bioinformatics workflows. However, big-data poses several challenges in these approaches. One of the challenges is how to handle the movement of big-data during workflow execution. With the advent of big-data, a number of modified orchestration and choreography approaches have also been developed to handle big-data. In this paper, we review and make a comparative assessment of the state-of-the-art approaches to execute big-data workflows. We examine the big-data handling in these approaches and finally recommend a solution that could be a way forward in executing big-data bioinformatics workflows.

**Keywords:** Big-data · Bioinformatics · Workflows · Orchestration · Choreography

## 1 Introduction

There is a big-data explosion in the field of bioinformatics, with the rapid growth in the size of biological data. Bioinformatics experiments typically involve analyzing data from one or more biological data sources using one or more analysis tools [1]. The biological data sources and the analysis tools are usually distributed and available as web services. Hence, workflows are used to integrate these distributed bioinformatics resources. Bioinformatics workflows are collection of analysis tool and data services combined in a certain way to represent a bioinformatics experiment. Each analysis tool or computation service in the workflow does some data analysis based on the input that it receives and produces some data based on the analysis for the next service. Bioinformatics workflows are executed based on data-flow. However, they can be executed based on control-flow or combining both data-flow and control-flow [2]. A data-flow describes the

A. Unal et al. (Eds.): SmartCom 2016, CCIS 628, pp. 647–654, 2016.
DOI: 10.1007/978-981-10-3433-6_78

flow of specific dataset among the services of the workflow, whereas the control-flow describes the correct order of execution among the different services of the workflow.

In literature, there are two workflow execution approaches i.e. the orchestration and the choreography approach [3]. In the orchestration approach, the workflow is executed using a central coordinator called the orchestrator or workflow engine. The orchestrator controls the data-flows and the control-flows among the services of the workflow. On the other hand, in choreography approach, the workflow services interact with each other directly to execute the workflow. The services share the control among themselves. Each service of the workflow knows with whom to interact and when to carry out its operations. Both these approaches are used to execute bioinformatics workflows.

Bioinformatics workflows are inherently complex. Their execution requires monitoring, reporting of workflow execution progress, handling of failures and recording of provenance data. Bioinformatics workflows are also data-intensive. They produce and move large volumes of data during workflow execution [4]. The complexity of bioinformatics workflow increases with big-data in it. Big-data pose several challenges in bioinformatics workflow execution. One of the challenges is how to handle the movement of big-data in bioinformatics workflows. With the advent of big-data, a number of modified approaches based on orchestration and choreography approaches have also been developed to handle the big-data and improve the performance of workflow execution. In this paper, we review and make a comparative assessment of the state-of-the-art approaches to execute big-data bioinformatics workflows. We examine the big-data handling in these approaches and finally recommend a solution that could be a way forward in executing big-data bioinformatics workflows.

## 2 Classical Approaches: Orchestration and Choreography

In this section, we review the classical orchestration and the classical choreography approaches. In order to understand the execution of bioinformatics workflows using these approaches, let us consider a workflow in the Fig. 1(a) with two services $s_1$ and $s_2$. Let *in* be the input from the user to service $s_1$. Based on this input *in*, $s_1$ produces the data $d$ that is passed on to service $s_2$. Service $s_2$ then produces the final output *out* which is passed to the user. The execution of the workflow using orchestration and choreography approaches is shown in the Figs. 1(b) and (c) respectively.

In the classical orchestration approach Fig. 1(b), the input *in* received from the user is sent to service $s_1$ by the orchestrator. Upon the receipt of the input *in*, the service $s_1$ produces the data $d$ which it passes to the orchestrator. The orchestrator then passes the data $d$ to the service $s_2$ which produces the final output *out*. The service $s_2$ passes this final output *out* to the orchestrator who forwards it to the user. In this approach all the data is routed among the services of the workflow through the central coordinator i.e. the orchestrator.

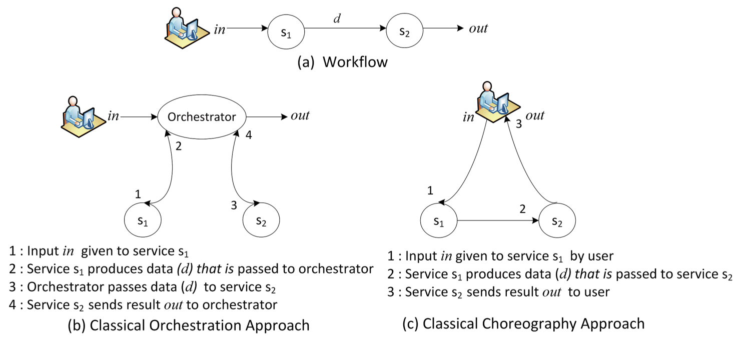
in
s1
d
s2
out
(a) Workflow
in
Orchestrator
out
1
2
3
4
s1
s2
1 : Input in given to service s1
2 : Service s1 produces data (d) that is passed to orchestrator
3 : Orchestrator passes data (d) to service s2
4 : Service s2 sends result out to orchestrator
(b) Classical Orchestration Approach
in
out
1
2
3
s1
s2
1 : Input in given to service s1 by user
2 : Service s1 produces data (d) that is passed to service s2
3 : Service s2 sends result out to user
(c) Classical Choreography Approach

**Fig. 1.** (a) Workflow (b) Classical orchestration approach (c) Classical choreography approach

Using orchestration in bioinformatics workflow has several advantages; this includes controlled and monitored execution of workflow, collection of provenance data and handling of task failures during workflow execution. However the classical orchestration approach suffers from performance bottlenecks. This is due the fact that all the data is indirectly passed between the services of the workflow resulting in unnecessary data-flows, consuming more bandwidth and hence weakening the performance of the workflow execution. Another issue with this approach is the scalability [5]. The overall performance of the bioinformatics workflow reduces as the number of services and the volume of the data to be orchestrated in the workflow becomes larger. The services of the workflow are distributed but the decision and coordination logic are centrally located at one point, this also creates a single point of failure in this approach.

In the classical choreography approach Fig. 1(c), the input *in* is given to the first service of the workflow i.e. the service $s_1$ by the user. The service $s_1$ produces the data $d$. This data $d$ is passed directly to service $s_2$ which produces the final output *out* which is transfered to the user. In a choreography approach the services of the workflow do not depend on the central coordinator. The approach achieves the composition by peer-to-peer communication between the services of the workflow [6].

The choreography approach facilitates the services of the workflow to transfer data directly between them without going through any central coordinator [7]. This direct data transfer between services consumes less bandwidth and thus provides performance benefits in bioinformatics workflow which are data-intensive. Further more the choreography approach is more scalable than orchestration approach [8]. However in this approach it is difficult to handle provenance of data, monitor and handle components failures. The choreography approach also suffers from implementation challenges i.e. the approach require complex design processes and execution infrastructure compared to orchestration approach. These difficulties make the orchestration as the preferred choice to execute bioinformatics workflows.

**Table 1.** Summary of orchestration and choreography approaches

| Factors | Method | |
|---|---|---|
| | *Orchestration* | *Choreography* |
| Composition | Centralized | Decentralized |
| Data movement | Indirect/Centralized | Direct/Peer-to-Peer |
| Monitoring | Easy | Difficult |
| Provenance | Easy | Difficult |
| Failure-handling | Relatively easy | Difficult |
| Scalability | Not scalable | Scalable |
| Single point of failure | Possible | Not possible |
| Implementation | Simple and straight-forward | Complex |

Table 1 summarises the orchestration and choreography approaches. As see from the Table 1, both the approaches have their share of advantages and disadvantages in executing bioinformatics workflows. In order to combine the benefits of both the approaches, several authors proposed modified orchestration and choreography approaches to handle big-data in workflow execution.

## 3 Modified Orchestration and Choreography Approaches

This section reviews the modified classical orchestration and choreography approaches. They are mostly based on decentralized and hybrid design. In a decentralized design both the control-flow and the data-flow are distributed. Whereas in a hybrid design the control-flow is centralized and the data-flow is distributed.

Binder et al. proposed a solution using *triggers* in [9]. A trigger is a lightweight infrastructure placed between the orchestrator and the services of the workflow. Each service is associated with a trigger, which invokes that service. Triggers act as data buffers and collect the required data for the services of the workflow before invoking the service. They forward the outputs to other triggers according to the routing information without involving the orchestrator for data transfers. The approach uses choreography to achieve decentralized control for workflow execution and overcomes the problem associated with the classical orchestration approach. Although the approach improves performance of the orchestrator, the data now moves between the triggers and the services, thereby increasing the traffic between them.

Barker et al. in [10] proposed a *proxy* based approach. This is a hybrid approach that combines the benefits of orchestration and the choreography approaches. A proxy is a lightweight piece of middleware placed between the orchestrator and the services of a workflow, controlled by orchestrator. The orchestrator sends a request to a proxy and the proxy then invokes the service on behalf of the orchestrator. The proxies buffer the intermediate data and pass it to the other proxies according to the data flow in the workflow. This allows the services of the workflow to pass that data among themselves without passing through the orchestrator. The proxies however pass references of the data passed between the services to the orchestrator. This allows the orchestrator to monitor the progress of workflow execution.

In [11], Fleuren et al. proposed a hybrid model that makes use of orchestrator and choreography approaches to execute workflow. The orchestrator is used to execute the main workflow and to integrate sub-workflows called *workflow skeletons* of the main workflows. While the choreography approach is used to execute the workflow skeletons. This approach also makes use of proxies, which are associated with each workflow skeletons to execute the workflow. Although proxies relieve the orchestrator from handling the intermediate data, the data now flows between proxies and services of the workflow increasing the data traffic between them.

Bahman Javadi et al. in [12] proposed decentralized orchestration approach based on *cloud* to execute workflows. The approach uses a cloud between the orchestrator and the services of the workflow to store and process data. Here the output data of the service invocation which is usually of large volumes are directly moved to the cloud bypassing the centralized orchestrator. In this way the orchestrator is relieved from sending the intermediate data and thus reducing the traffic between the services and the orchestrator. Since cloud provides high amount of storage and processing, this approach can handle much more data then the triggers and proxies approach. However, this approach directs all the data from the services to the cloud, thus increasing the traffic between services and the cloud.

Ward Jaradat et al. in [13] also proposed a cloud based workflow execution approach that makes use of *distributed orchestration*. This decentralized orchestration approach partitions the workflow into smaller sub-workflows and then transmits these sub-workflows to appropriate cloud location and then their execution happens in parallel. At each location an orchestrator is used to coordinate the sub-workflow execution. The distributed orchestrators transfer intermediate results between them to complete the workflow execution. Since the workflow is executed in parallel the performance of workflow execution improves. Although clouds provide advantages in execution of bioinformatics workflows in terms of large storage space and computing facilities, one of the big challenge faced on the cloud is handling the movement of the large data sets in and out of the cloud.

Wieland et al. in [14] proposed the concept of *pointers* to pass data by reference rather than by value from one service to another service of the workflow. The service that produces the data, transfers a reference of data to consuming service via the orchestrator. The consuming service then uses this reference to get the data from shared data storage. The orchestrator only has to handle the reference and not the data. Thus, this mechanism reduces the load on the orchestrator leading to faster workflow execution. However, the data moves between the service which increases the communication cost between the services especially when the data is too big to move.

**Table 2.** Overview of the modified approaches

| | | | | | Data-flow | |
|---|---|---|---|---|---|---|
| Approach Name | Design | Interface | Environment | Single point of failure places | Type | Between |
| Triggers [9] | Decentralized | Mediator | SOA | Orchestrator/ Triggers | Indirect | Triggers-and-Services |
| Proxy [10] | Hybrid | Mediator | SOA | Orchestrator/ Proxies | Indirect | Proxies-and-Services |
| Workflow Skeletons [11] | Hybrid | Mediator | SOA / Grid | Orchestrator/ Proxies | Indirect | Proxies-and-Services |
| Cloud [12] | Decentralized | Mediator | SOA / Cloud | Orchestrator | Indirect | Services-and-Cloud |
| Distributed Orchestration [13] | Decentralized Orchestration | Distributed Orchestrators | SOA / Cloud | Distributed Orchestrators | Indirect | Distributed Orchestrator-and-Services |
| Pointers/ Reference passing [14] | Decentralized | Mediator | SOA | Orchestrator | Indirect | Shared data storage-and -Services |
| Data Flow Delegated (DFD) [15] | Hybrid | Direct/ Peer-to-Peer | SOA | Orchestrator | Direct | Services |
| Pipelined Data Flow Delegated (PDFD) [16] | Hybrid | Direct/Peer-to-Peer | SOA | Orchestrator | Direct | Services |

Subramanian et al. proposed a *Data-flow delegated* (*DFD*) approach in [15]. The data required in the workflow execution are dynamically assigned to the workflow services as per their requirements. This is achieved by enabling direct transport of data between participating services of the workflow controlled by the orchestrator. The orchestrator informs the services from where they will receive the input data and where they have to send the output data. The Data-Flow Delegated (DFD) approach relieves the orchestrator from handling the data but the services of the workflow move the data directly between them.

Although pipelined parallelism is used in workflow systems such as Kepler, Taverna and Triana, they carry the problem associated with the orchestration i.e., they require the orchestrator to transmit (and receive) all the input and output data of the component services. Subramanian et al. attempted to overcome this problem in [16]. The authors proposed an approach called *Pipelined data-flow delegation (PDFD)* for web services-based workflows. Pipelined data-flow delegation is orchestrator coordinated approach that allows partitioning of large datasets into independent subsets and that can be communicated in a pipelined manner without going through the centralized orchestrator. This approach improves workflow execution but is feasible only if data can be split into independent chunks and processed in batches.

## 4 Discussion and Conclusion

Table 2 gives an overview of the modified approaches reviewed in this paper. Most of the approaches are based on decentralized and hybrid design. The approaches handle big-data in a workflow by moving the data either directly between the services or indirectly using mediators and data references. Parallelism is also combined in some of the approaches to handle big-data and to speedup workflow execution.

The modified approaches are data-driven. They move data which is varying in size from one computing service to another computing service during the workflow execution. The orchestrator is used to control and monitor the workflow execution, while the computing services of the workflow handle the data-flow directly or indirectly between them. In other words the data coordination responsibility of the orchestrator is distributed to the workflow components. However, such distribution can optimize the workflow performance to some extent, but not extensively as the distribution of responsibilities does not help to overcome all the difficulties in handling big-data. With the size of biological data increasing and the workflows having to handle data in the range of terabytes to petabytes and more, moving data to computation in a workflow is not feasible solution. The size of the analysis tools associated with computing services are much smaller than the size of the data that flows in a workflow. To handle the big-data in a bioinformatics workflow, this paper suggests changing the paradigm of workflow execution by moving the computation to data. Moving computation means moving analysis tools from computation services to data services. We feel this will lead to more efficient handling of big-data in bioinformatics workflows and thereby mark a shift in big-data analysis.

## References

1. Stevens, R.D., Tipney, H.J., Wroe, C.J., Oinn, T.M., Senger, M., Lord, P.W., Goble, C.A., Brass, A., Tassabehji, M.: Exploring Williams-Beuren syndrome using myGrid. Bioinformatics **20**(suppl 1), i303–i310 (2004)
2. Yang, X., Wang, L., Jie, W. (eds.): Guide to e-Science: Next Generation Scientific Research and Discovery. Springer, Heidelberg (2011)
3. Barker, A., van Hemert, J.: Scientific workflow: a survey and research directions. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 746–753. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). doi:10.1007/978-3-540-68111-3_78
4. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. J. Grid Comput. **13**(4), 457–493 (2015)
5. Barker, A., Besana, P., Robertson, D., Weissman, J.B.: The benefits of service choreography for data-intensive computing. In: Proceedings of the 7th International Workshop on Challenges of Large Applications in Distributed Environments, pp. 1–10. ACM, June 2009
6. Barker, A., Weissman, J.B., Van Hemert, J.: Eliminating the middleman: peer-to-peer dataflow. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 55–64. ACM, June 2008
7. Barker, A., Walton, C.D., Robertson, D.: Choreographing web services. IEEE Trans. Serv. Comput. **2**(2), 152–166 (2009)
8. Pedraza, G., Estublier, J.: Distributed orchestration versus choreography: The FOCAS approach. In: Wang, Q., Garousi, V., Madachy, R., Pfahl, D. (eds.) ICSP 2009. LNCS, vol. 5543, pp. 75–86. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01680-6_9

9. Binder, W., Constantinescu, I., Faltings, B.: Service invocation triggers: a lightweight routing infrastructure for decentralised workflow orchestration. Int. J. High Perform. Comput. Networking **6**(1), 81–90 (2009)
10. Barker, A., Weissman, J.B., van Hemert, J.I.: The circulate architecture: avoiding workflow bottlenecks caused by centralised orchestration. Cluster Comput. **12**(2), 221–235 (2009)
11. Fleuren, T., Götze, J., Müller, P.: Workflow skeletons: increasing scalability of scientific workflows by combining orchestration and choreography. In: IEEE European Conference on Web Services (ECOWS), pp. 99–106, September 2011
12. Javadi, B., Tomko, M., Sinnott, R.O.: Decentralized orchestration of data-centric workflows in cloud environments. Future Gener. Comput. Syst. **29**(7), 1826–1837 (2013)
13. Jaradat, W., Dearle, A., Barker, A.: Workflow partitioning and deployment on the cloud using orchestra. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 251–260. IEEE Computer Society, December 2014
14. Wieland, M., Görlach, K., Schumm, D., Leymann, F.: Towards reference passing in web service and workflow-based applications. In: IEEE International Enterprise Distributed Object Computing Conference, pp. 109–118. IEEE, September 2009
15. Subramanian, S., Sztromwasser, P., Petersen, K., Puntervoll, P.: Direct data transfer between SOAP web services in orchestration. In: Proceedings of the 14th International Conference on Information Integration and Web-based Applications and Services, pp. 91–100. ACM, December 2012
16. Subramanian, S., Sztromwasser, P., Puntervoll, P., Petersen, K.: Pipelined dataflow delegated orchestration for data-intensive eScience workflows. Int. J. Web Inf. Syst. **9**(3), 204–218 (2013)