Direct data transfer between SOAP web services in Orchestration

Sattanathan Subramanian
Uni Computing, Uni Research, Bergen, Norway
sat@uni.no

Paweł Sztromwasser
Department of Informatics, University of Bergen, Norway
Uni Computing, Uni Research, Bergen, Norway
pawels@uni.no

Pål Puntervoll
Uni Computing, Uni Research, Bergen, Norway
pal.puntervoll@uni.no

Kjell Petersen
Uni Computing, Uni Research, Bergen, Norway
kjell.petersen@uni.no

ABSTRACT

In scientific data analysis, workflows are used to integrate and coordinate resources such as databases and tools. Workflows are normally executed by an orchestrator that invokes component services and mediates data transport between them. Scientific data are frequently large, and brokering large data increases the load on the orchestrator and reduces workflow performance. To remedy this problem, we demonstrate how plain SOAP web services can be tailored to support direct service-to-service data transport, thus allowing the orchestrator to delegate the data-flow. We formally define a data-flow delegation message, develop an XML schema for it, and analyze performance improvement of data-flow delegation empirically in comparison with the regular orchestration using an example bioinformatics workflow.

Keywords: Orchestration, Bioinformatics Workflow, Web service, and Workflow Management.

1. INTRODUCTION

The complex requirements of scientific data analysis and knowledge discovery can rarely be satisfied by a single database or a tool [15]. Workflows originating from business applications have been considered to be a solution satisfying these requirements. By integrating and coordinating geographically separated resources, workflows automate scientific analysis in a structured and controlled manner. Scientific workflows differ from its business predecessors, mostly in being data-oriented, i.e., "data is considered as a first-class citizen" [21]. The two widely proposed approaches for executing workflows are orchestration (also known as *centralized coordination*) and choreography (also known as *decentralized coordination*). The orchestration approach, as used by BPEL [3], requires a centralized coordinator (i.e., orchestrator) to establish the control and communication between two or more resources. In contrast, the choreography allows peer-to-peer communication between activities without a separate coordinator, as given in e.g., WS-CDL[1]. Most of the scientific workflow engines use orchestration to execute workflows, e.g., Taverna [22]. There are several reasons for this, including ease in handling component service failures [16]; in controlling [4], monitoring [18] and validating [10] workflow executions; in collecting the history of a workflow execution [4]; in reusing the existing open-source (e.g., ActiveBPEL[2]) and industry (e.g., Microsoft WinWF[3]) based workflow engines, and last but not the least, the lack of choreography supportive engines and services. For these reasons and despite the fact that the choreography is proposed as a more efficient method for data-oriented workflows, choreography-based methods for executing workflows (e.g., MAP [5]) have not gained wide acceptance and popularity in the scientific community.

The web service technology is advocated as a convenient solution to expose various scientific resources (i.e, tools and databases) to the web [14], also in the field of bioinformatics [19] which has our prime interest. Bioinformatics focuses on developing algorithms and tools for analysis of biological data. Integral parts of the field are also storage, retrieval, and integration of the data and the tools. Using web services, bioinformaticians can *describe* the tools they develop in a machine-readable format, *publish* them into one or more registries, *discover* and *access* tools developed by other groups, and *compose* complex workflows, if a single resource doesn't meet scientific requirements. For the last couple of years, many of the bioinformatics data and tool providers have been publishing their resources in the form of programmatically accessible web services. BioCatalogue provides a curated catalogue of life science web services having 2319 services[4] from 169 providers, in which the majority of services are SOAP web services [7], i.e., web services de-

[1] http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/
[2] http://www.activevos.com/community-open-source.php
[3] http://msdn.microsoft.com/en-us/netframework/aa663328.aspx
[4] visited on 13th Aug 2012

scribed by WSDL[5] and communicating using SOAP[6] over HTTP[7] protocol. In line with this technological evolution, bioinformatic resources are considered SOAP web services in this paper.

In bioinformatics, workflows often process large amounts of data, e.g. DNA sequencing experiment produces tens of gigabytes of data for every sequenced person. The centralized coordination requires that the orchestrator sends and receives the input and output data of the component web services, in addition to the services invocation. Mediating the large data-transfer between the component services increases use of the coordinator's resources excessively (e.g., memory, bandwidth), and increases the workflow execution time. In [20], we have proposed an approach called *Data-Flow Delegation* to overcome this data-flow bottleneck. In that paper, we have given a theoretical foundation, for separating the control and the data-flow in a workflow execution in order to delegate the data-flow responsibilities to the component services. The data-flow delegation happens on the fly according to the workflow requirements, i.e., the coordinator instructs the component services where the input data will be sent from, and where to send the output data. In this paper, we show that the data-flow delegation proposed in our previous paper [20] can be applied to delegate data between plain SOAP web services. We used concept of operation overloading to inject data-flow delegation instructions into service requests. In consequence the changes in the service interface do not break existing clients and allow use of the service in a regular manner. Our approach was implemented on top of a web service framework, showing that it is independent of the framework and does not require any changes to it. Our contributions in this paper include:

(i) an XML schema developed to support data-flow delegation between plain SOAP web services, and used to overload web-service operations

(ii) empirical analysis showing performance improvement of the data-flow delegating workflow in terms of memory, CPU, network traffic, and runtime

The organization of the remaining paper is as follows: Section 2 reports the related work available in the literature for overcoming data-flow bottleneck of orchestrator. Section 3 provides a background about the regular and the data-flow delegated orchestration. Section 4 presents an usecase from the domain of bioinformatics. Section 5 explains the concept of implementation including the formal definition of data-flow delegation message. Section 6 shows the workflow execution results and analyzes them in detail. Finally, Section 7 concludes the paper and outlines our future work.

2. RELATED WORK

Several approaches have been proposed to overcome the data-flow bottleneck of the orchestrator. We classify them into three categories and list those here.

Introducing a middle-layer between orchestrator and component services. Triggers and proxies are suggested as data-brokers between orchestrator and component services in [8] and [6], respectively. They are responsible for buffering the intermediate data and for invoking the component services on behalf of the coordinator. Authors of [13] propose to use a cloud infrastructure as a service in between the orchestrator and component services, similarly to the proxy approach. Since the cloud can offer high amount of storage and processing, this broker can handle much more data than proxies and triggers. In [1, 11], a shared data storage is proposed for sharing the intermediate data between the component services. A clear advantage of the above approaches is that the proposed middle layer, that brokers the data-flow on behalf of the orchestrator, relieves the orchestrator from sending and receiving the intermediate data. On the other hand it restricts the coordinator to fully rely on the brokers for communicating with the component services. For example, a coordinator cannot invoke a component service when a proxy is down. The data to and from a component service always flows through a broker, which can potentially reduce the performance of the workflow. Moreover, since the brokers are separate entities from the component services, those will require additional maintenance and monitoring in addition to the development.

Using stateful web services to pass data as resources. Seiler et al. in [17] and Heinzl et al. in [12] proposed to pass data as a reference between services through orchestrator, i.e., after receiving a data reference from a producer service, the coordinator forwards the reference to a consumer service. The consumer service is responsible for getting the data using the reference. The idea was realized using WSRF[8] resources. Zhang et al. [24] came up with a similar approach suggesting to forward the intermediate data as WSRF resources directly from one web service to another in the process of orchestration. It is realized by extending an existing WSRF-compliant web services framework with data-forwarding capabilities. Although both solutions are very elegant they are not easily applicable to bioinformatics, where the WSRF services are rarely used. Lack of support for WSRF in open-source web service stack implementations is one of the reasons for it. Use of WSRF services has also been discouraged in the bioinformatics community, for instance by the EMBRACE Technology Recommendations[9].

Upgrading component services to workflow engines. In a different proposal in [23], component services are upgraded to workflow engines. They can identify and perform their tasks (including data transfer) by examining the workflow description (i.e., script), like in choreography. Unfortunately, not only the flow of data is delegated from one service to another at runtime, but also the control. This contrasts with the most of scientific workflow systems where the orchestrator is required to keep the control for retaining the advantages of centralized coordination, i.e., monitoring, validating, fault-handling, and provenance.

[5] `http://www.w3.org/TR/wsdl`
[6] `http://www.w3.org/TR/soap/`
[7] `http://www.w3.org/Protocols/`
[8] Web Services Resource Framework `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`
[9] `http://www.embracegrid.info`

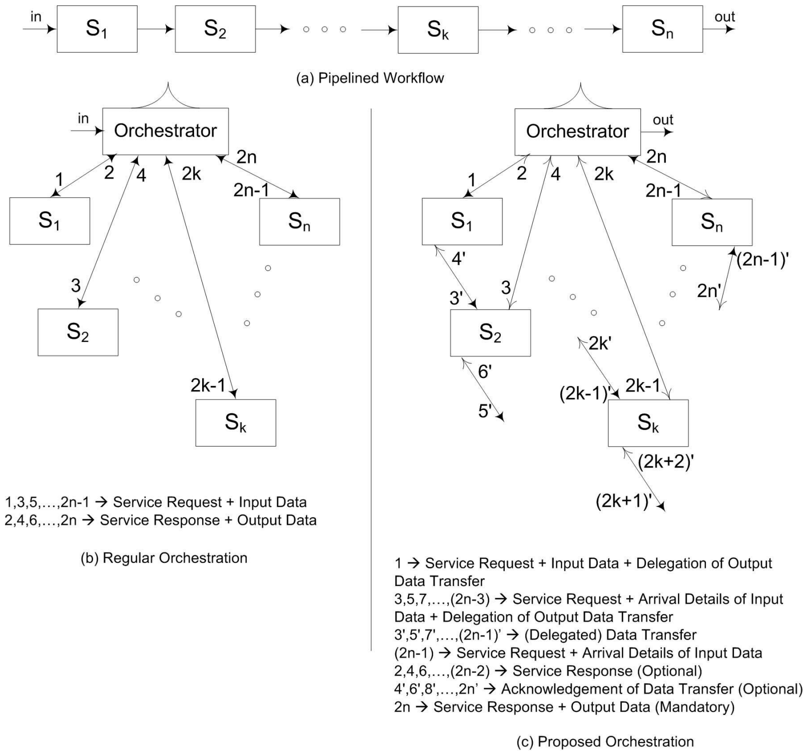

Figure 1: (a) A Pipelined Workflow, (b) Regular Orchestration, and (c) Data-flow Delegated Orchestration

3. BACKGROUND: REGULAR AND DATA-FLOW DELEGATED ORCHESTRATION

We assume that a DAG (Directed Acyclic Graph) workflow is limited by the slowest (or most resource-demanding) path of services through the graph. Such a path of services in a DAG is a pipelined sub-part of the workflow. Improving performance of the pipelined sub-part elevates in effect overall performance of the workflow. Thus we focus on improving pipelined workflows, and expect that the results can easily be transferred on any DAG workflow.

Figure 1(a) presents a pipelined workflow used for illustrating the execution style of the regular and the data-flow delegated orchestrations. The pipelined workflow w consists of n number of services, i.e., $s_1, s_2, ..., s_n$. The service s_1 begins the workflow w receiving the initial input *in*. After processing, the output of s_1 is send as input to service s_2. Likewise, the output of service s_{k-1} is the input to service s_k, where, $2 \leq k \leq n$. The service s_n ends the workflow and sends out the final output *out*.

Figure 1(b) shows the regular approach of orchestration used to execute workflow w, given in Figure 1(a). The orchestrator invokes service s_1 and feeds the respective input data (*1*). After processing the request, service s_1 provides the response including the output data to the orchestrator (*2*). Next, the orchestrator invokes service s_2 with the input data received from service s_1 (*3*). After processing this request, s_2 delivers the output data with the response to the orchestrator (*4*). In general, the orchestrator supplies the output data of service s_{k-1} to service s_k (where, $k \in \{2, 3, ..., n\}$) as the input data during the invocation of service s_k (*2k-1*), and service s_k returns the response with the output data to the orchestrator (*2k*).

Figure 1(c) shows the execution of data-flow delegated orchestration. The orchestrator invokes service s_1 along with the respective input data, and delegates the responsibility

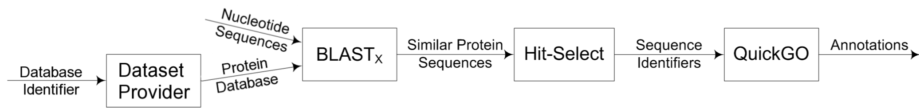

Figure 2: Simple Gene Annotation Workflow

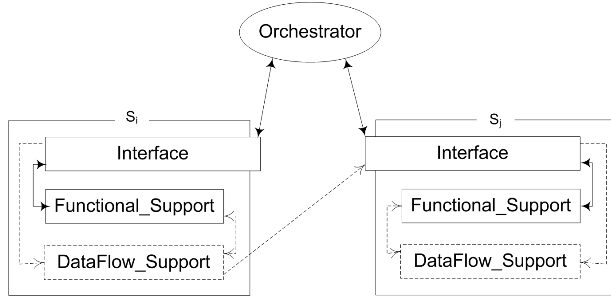

Figure 3: For Delegated Data-flow between Services

to transmit the output data directly to service s_2 (*1*). The orchestrator directly supplies the input data to service s_1, since service s_1 begins the processing of workflow w. Service s_1 provides the service response to the orchestrator, without the output data (*2*). The orchestrator invokes service s_2 with the instruction to wait for the input data from service s_1, and delegates the responsibility to transmit the output data directly to service s_3 (*3*). Service s_2 receives the input data from service s_1 (*3'*), provides the service response to the orchestrator without the output data (*4*), and acknowledges service s_1 for transmitting the input data (*4'*). In general, the orchestrator invokes service s_k (where, $k \in \{2, 3, ..., (n-1)\}$) with the instruction to wait for the input data from service s_{k-1}, and delegates the responsibility to transmit the output data directly to service s_{k+1} (*2k-1*). During the invocation of service s_n, the orchestrator does not delegate the flow of output data to any other service since service s_n ends the processing of workflow w. So, service s_n transmits the output data to the default target, i.e., to the orchestrator which invokes the service (*2n*). Similar to service s_k, service s_n receives the input data from service s_{n-1} ((*2n-1*)') based on the direction given by the orchestrator, and acknowledges service s_{n-1} for transmitting the input data (*2n'*). The delegation of data-flow relieves the orchestrator from receiving the intermediate data results from services s_1 to s_{n-1}, and sending those data to services s_2 to s_n in the respective order.

4. USE CASE

Bioinformatics tools are used to perform many of the key steps involved in characterizing the genomes of living organisms: the short experimentally produced DNA sequence reads are assembled into *contigs* that represent partial or complete genomic sequences; the gene structures of those sequences are predicted; and finally the identity and function of the individual genes are predicted. We have chosen the last step of this process, which is frequently referred to as *gene annotation*, to illustrate a bioinformatics workflow. Gene annotation exploits the fact that all organisms and their genes are related through the process of evolution. This implies that if a newly sequenced gene has a high degree of sequence similarity to a gene of known function from a different species, it is likely to have the same or a similar function. Figure 2 shows a simplified way of predicting functions in the newly sequenced genes. The DNA sequence of each gene is searched against a database (the first step of Fig. 2) of manually curated protein sequences (UniProt/SwissProt[10]) using the BLAST [2] sequence similarity search program provided by the `BLASTx` service (second step of Fig. 2). The BLAST result is then passed on to the `Hit-Select` service (third step of Fig. 2) that extracts the sequence identifier of the best protein sequence hit. Finally Gene Ontology terms [9], describing the functions of genes/proteins, are fetched from the `QuickGO`[11] service (fourth step of Fig. 2), and returned as functional annotations of the newly sequenced genes.

5. IMPLEMENTATION CONCEPT

5.1 Service Operation Overloading

Figure 3 shows the delegated data-flow between the services s_i and s_j, and their supportive functions. Existing elements of the services are shown with normal-lines, and the proposed elements are shown with dotted-lines. The existing

[10] `http://www.ebi.ac.uk/uniprot/`

[11] `http://www.ebi.ac.uk/QuickGO/WebServices.html`

elements of a service are an *Interface* that allows for access to the service, and the *Functional_Support* that provides the functionality of the service. The communication between the *Interface* and *Functional_Support* is internal and bidirectional. Operations in the *Interface* have `FunctionalArguments` that are required to execute service's functionality. To enable data-flow delegation we propose to overload operations in the service's *Interface* with an optional argument—`DF-Delegation`. The purpose of the `DF-Delegation` parameter is to instruct the service to (i) wait for its input data from a given source; (ii) send its output data to a given destination. The details of the `DF-Delegation` parameter are given in Definition 1. The overloading enables the service invoker to access the same *Interface* of the service for satisfying three different aims:

(i) To receive the functionality of the service without delegating the flow of data, e.g., the orchestrator makes the service request with the input data and receives the service response with the output data. The `FunctionalArguments` are used for this.

(ii) To send the delegated data without requiring the functionality of the service, e.g., the service s_i transfers its output data to the service s_j as required by the orchestrator. The `DF-Delegation` parameter is used for this.

(iii) To receive the functionality of the service with the delegation of data-flow, e.g., the orchestrator makes the service request to s_i with the instruction to wait for input data (or transfer the output data) from (or to) s_j. This requires both parameters, i.e., `FunctionalArguments` and `DF-Delegation`.

The *DataFlow_Support* is the proposed element for supporting the data flow between services according to the delegation instruction (`DF-Delegation`) provided by the orchestrator. As shown in Figure 3, the *DataFlow_Support* should have the capability to: (i) invoke an external service $interface$ for transferring the ouput data according to the requirements of data-flow delegation; (ii) receive the input data from another service through the local service $interface$; and (iii) interact with the *Functional_Support* for providing the received input data. The transfer of delegated data can optionally end with an acknowledgement from the recipient service.

Definition 1 ($\mathcal{DF}$**-Delegation**). *A* $\mathcal{DF}$*-Delegation is a 4-tuple* $\mathcal{D}_{df} = \langle C, W, Data_in, Data_out \rangle$ *where:*

- *C is a description of the orchestrator. It is a 2-tuple* $\langle c_name, c_id \rangle$*, where, c_name is the name of orchestrator, and c_id is an identification of orchestrator.*

- *W is a description of the workflow. It is a 2-tuple* $\langle w_name, w_id \rangle$*, where, w_name is the name of workflow, and w_id is the identification of workflow.*

- *Data_in contains the details for receiving and recognizing the data from another service of a workflow. It is a 5-tuple* $\langle s_id, d_id, d_type, d_size, v_time \rangle$*, where, s_id is the identification of service from which the input data comes, d_id is the identification of data, d_type is the data type, d_size is the data size, and v_time is the time validity of data. In this, d_id and s_id are mandatory, and the remaining are optional.*

- *Data_out provides the details for sending out the data. It is a 5-tuple* $\langle s_id, s_link, d_id, d_type, a_time \rangle$*, where, s_id is the identification of the service which requires the output data, s_link is a URL of the data receiver in s_id, d_id is the identification of the data, d_type is the data type, and a_time is the time allocated for sending the data to the service which has the identification s_id. In this, s_id, s_link, and d_id are mandatory, and the remaining are optional.*

5.2 XML Schema to Delegate Data-Flow

The input messages of the WSDL operations were extended (overloaded) with a new element to support the data-flow delegation. A fragment of the WSDL from the `Hit-Select` service is given in Listing 1. The snippet contains definition of an input message to the *selectBest* operation. The message contains two elements: *BlastOutput* and *dataDelegation*. The *BlastOutput* is the functional argument of the operation, i.e. the input data. The *dataDelegation* element has been added to enable data-flow delegation. The argument is optional allowing invocation of the operation without data-flow delegation. The *dataDelegation* element is of type *DFDelegation* and contains all the information that is required to delegate flow of the data, as specified in Definition 1. XML schema for the *DFDelegation* type representing the $\mathcal{DF}$-$\mathcal{D}$`elegation` tuple is presented in Listing 2.

Listing 1: An Operation from the `Hit-Select`

```
<xsd:element name="selectBestRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="BlastOutput" type=
          "blast:BlastOutputType" minOccurs
          ="0"/>
      <xsd:element name="dataDelegation"
          type="dd:DFDelegation" minOccurs=
          "0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
  ...
<wsdl:message name="selectBestRequest">
  <wsdl:part element="tns:selectBestRequest
      " name="parameters"/>
</wsdl:message>
  ...
<wsdl:portType name="BestHitSelect">
  <wsdl:operation name="selectBest">
    <wsdl:input message="
        tns:selectBestRequest"/>
    <wsdl:output message="
        tns:selectBestResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

Listing 2: XML Schema of $\mathcal{DF}$Delegation

```
<schema targetNamespace="http://www.esysbio
    .org/dataDelegation" elementFormDefault
    ="qualified">
  <complexType name="DFDelegation">
    <sequence>
      <element name="coordinator" type="
          tns:CoordinatorType"/>
      <element name="workflow" type="
          tns:WorkflowType"/>
      <element name="dataIn" type="
          tns:DataInType" minOccurs="0"/>
        <element name="dataOut" type="
            tns:DataOutType" minOccurs="0"/
            >
    </sequence>
  </complexType>
  <complexType name="CoordinatorType">
    <sequence>
      <element name="id" type="string"/>
      <element name="name" type="string"/>
    </sequence>
  </complexType>
  <complexType name="WorkflowType">
    <sequence>
      <element name="id" type="string"/>
      <element name="name" type="string"/>
    </sequence>
  </complexType>
  <complexType name="DataInType">
    <sequence>
      <element name="serviceID" type="
          string"/>
      <element name="dataID" type="string"/
          >
      <element name="dataType" type="string
          " minOccurs="0"/>
      <element name="dataSize" type="int"
          minOccurs="0"/>
      <element name="validity" type="
          dateTime" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="DataOutType">
    <sequence>
      <element name="RecipientService" type
          ="tns:RecipientServiceType"/>
      <element name="dataID" type="string"/
          >
      <element name="dataType" type="string
          " minOccurs="0"/>
      <element name="allocatedTime" type="
          dateTime" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="RecipientServiceType">
    <sequence>
      <element name="serviceID" type="
          string"/>
      <element name="serviceName" type="
          string"/>
      <element name="serviceWSDL" type="
          anyURI"/>
      <element name="servicePort" type="
          string"/>
      <element name="serviceOperation" type
          ="string"/>
      <element name="delegatedArgumentName"
          type="string"/>
    </sequence>
  </complexType>
</schema>
```

6. EMPIRICAL ANALYSIS

Two versions of the *Simple Gene Annotation Workflow* shown in Figure 2 were implemented to compare the performance of the regular and the data-flow delegating approaches.

6.1 Technology, Hardware, Input, and Measured Factors

The technologies used in this implementation were the *Python v2.6.4* language and the *ZSI v2.1_a1*[12] library, for the development of a workflow `orchestrator` and four component web services, i.e., `Dataset Provider`, `BLASTx`, `Hit-Select`, and `QuickGO`. ZSI supports dynamic generation of the web services client code needed for satisfying the requirements of the data-flow delegation approach proposed in this paper. The Unix system monitoring tool *top* was used to monitor processor and memory usages, and the IBM developerworks' *nmon*[13] tool was used to assess network traffic usage. Both monitoring tools were invoked in an automated manner tracing resource utilization with 1 second intervals.

The orchestrator and the component web services were executed on separate Linux machines, sharing a 1Gbit/s network. The orchestrator and two of the component services (i.e., `BLASTx` and `Hit-Select`) were deployed on duo core 2.4GHz processor machines with 4GB of RAM, and the other component web services (i.e., `Dataset Provider` and `QuickGO`) were hosted on separate duo core 2.4GHz processor machines with 2GB of RAM.

Input to the *Simple Gene Annotation Workflow* (Fig. 2) were 10 sets of *nucleotide sequences*, used as query sequences in the BLAST search provided by the `BLASTx service`, and a *database identifier* used to specify which database of protein sequences to search against. The 10 sets of nucleotide sequences were sampled from a set of 400, and contain 1, 2, 4, 6, 12, 25, 50, 100, 200, and 400 nucleotide sequences, respectively. The data size of the subsets ranged from ∼1 KB to ∼550 KB. The protein sequence database used in all runs, and provided by the `Dataset Provider` service, was the UniProt/SwissProt database (Release 2010_06), which is a 230.7 MB text file containing 517,100 protein sequences. The size of the database is constant and does not change with the input size of the workflow.

The factors considered during the performance measurements were: *network traffic* (KB), *memory usage* (MB), *CPU usage* (CPU-seconds), and *run-time* (s).*Network traffic* is the sum of the data received and transmitted by the web service stacks of the coordinator and component web

[12] `http://pywebsvcs.sourceforge.net/`

[13] `http://www.ibm.com/developerworks/aix/library/au-analyze_aix/`

services. *Memory usage* is the sum of memory required by the web service stacks of the coordinator and component web services. We measured accumulated memory usage with one second intervals to emphasize also the time the memory is allocated for, and not only the peak memory use. *CPU usage* is the number of CPU-seconds consumed by the web service stacks of the coordinator and component web services. The CPU-seconds are calculated by summarizing the fraction of maximum CPU load multiplied by one second of measurement interval. *Run-time* is the time it takes to complete the full workflow execution, which is the time that elapses from the departure of the initial input data to the arrival of the final output data. Workflow runs were monitored by performing measurements every second. Each test was repeated five times to ensure reliable results.

6.2 Results and Analysis

The overall difference in performance between the two workflow approaches are shown in Fig. 4, and demonstrates that the *data-flow delegating* approach is superior to the *regular* approach for all considered performance factors.

The *network traffic* usage (Fig. 4(c)) increases with increasing data size for both approaches as expected, but the pace of the increase and almost identical difference between the approaches regardless of the input size warrants an explanation. The protein database transfer between the `Dataset Provider` service and the `BLASTx` service is responsible for a substantial part of the network traffic. The size of the database is constant (approx. 230MB) and does not depend on the input size, thus it has a larger effect on network traffic figures for smaller input sizes. The workflow delegating the data transfer can skip one transmission of all the intermediate data (including the large database), reducing network traffic by half, i.e., for data sets with 1 to 100 sequences the data-flow delegating workflow required 49.4%-49.7% less bandwidth than the regular workflow. For larger input sizes, the input and output of the workflow (which are not delegated) constitute a larger part of the total data traffic, so the percentage improvement achieved by delegating the intermediate data drops minimally, i.e., for 200 sequences it was 48.7%; and for 400 sequences it was 47.7%.

Memory, *CPU*, and *run-time* also increases with data size for both approaches, but unlike for *network traffic* the difference is negligible for small data sets and becomes significant with larger data. The CPU usage improvement for the data-flow delegating approach gradually increases from 7.6% to 37.9% for 1 to 50 sequences, and then it apparently levels off: 42.4% for 100 sequences; 44.7% for 200 sequences; and 44.9% for 400 sequences. This indicates that the delegating approach close to halves the workflow CPU usage for larger data sets. Differences in memory usage shows a slightly different pattern. The delegating workflow consumes around 25% less memory than the regular approach for data sets of 1 to 50 sequences. For larger data sets, the difference becomes more pronounced: 29.2% for 100 sequences; 37.0% for 200 sequences; and 52.8% for 400 sequences. The rapid growth of memory optimization for 100 and more sequences is caused by a shift in what the memory is used for. For the input size up to 50 sequences, the protein database (see, Figure 2) uses significantly larger part of memory than the input and output data. The memory requirement of protein

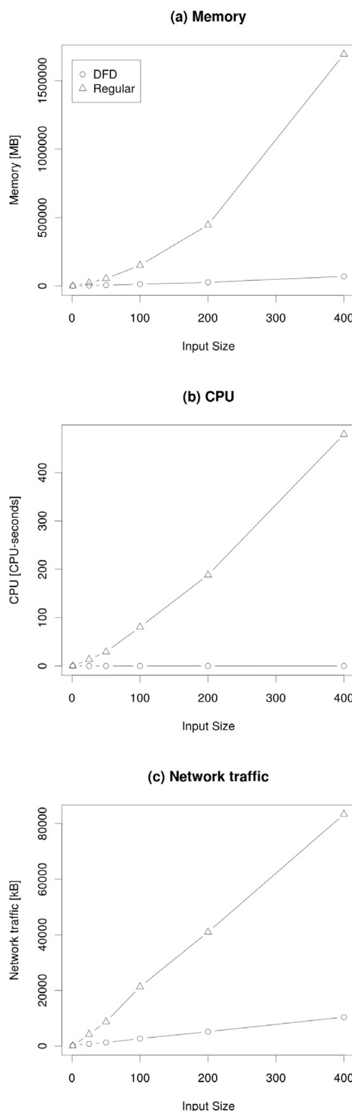

Figure 5: Performance Analysis of the *orchestrator*

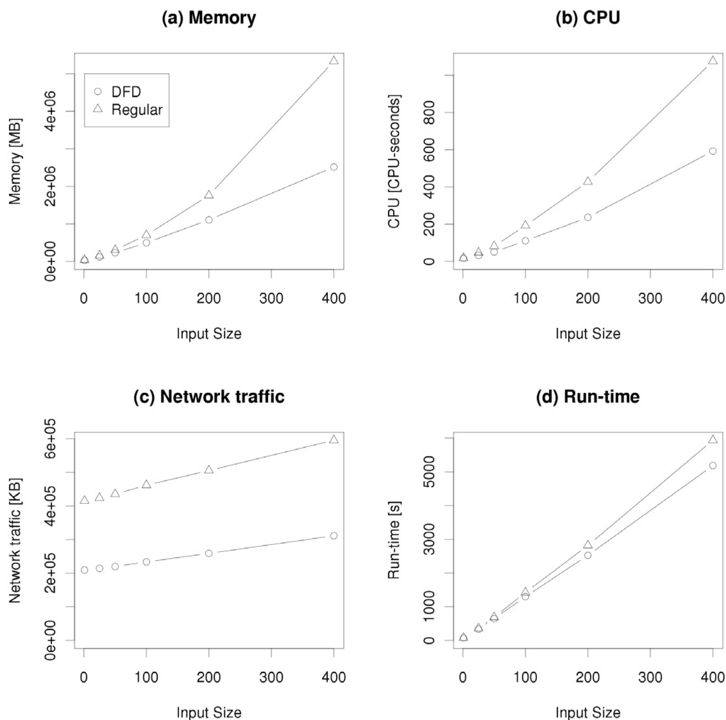

Figure 4: Performance Improvement for the Workflow.

database is constant (as its size), which keeps the percentage of optimization at around 25%. But from 50 sequences, the growing input, output and intermediate data use larger part of memory than the protein database. Therefore, we observe a rapid growth in memory optimization for large input data. This suggests that the delegating approach scales very well with regards to memory, and that it will significantly outperform the regular approach for large input data. The run-time of the two smallest data sets are slightly longer for the delegating approach than for the regular approach: 1.8% for 1 sequence; and 0.2% for 2 sequences. However, from 4 to 400 sequences, the delegating approach is faster, with reduced run-times ranging from 0.4% to 12.7%. The difference in run-time seems to level off, and is not expected to change significantly for larger data sets, in this case. It is because the workflow that we chose for this experiment requires far more time for computation than for the data-transfer, and communication optimizations have limited influence on the run-time improvement. We expect that the run-time improvement will be larger and it will be growing steadily according to the input size, when the data-transfer constitutes a larger share of the total workflow run-time.

The results described in the previous section clearly demonstrate that the delegating workflow outperforms the regular approach. However, to resolve the impact of the delegating approach on the `orchestrator` some of the performance measurements are presented in more detail. The `BLASTx` and `Hit-Select` services have both input and output data delegated, in contrast to the `Dataset Provider` and `QuickGO` services, where only the output or input data is delegated. In the following, we discuss only on the middle section of the workflow where the fully delegated services are invoked, i.e. between `BLASTx` and `Hit-Select` services. The coordinator is expected to be more severely affected when invoking these services with data-flow delegation, compared to the partially delegated ones. Also, the following analysis excludes the protein database transfer between the `Dataset`

Provider service and BLASTx service (see, Figure 2) to improve the clarity of results.

The performance details for the orchestrator is shown in Fig.5. The immediate observation one can make is that the data-flow delegation has a significant impact on the orchestrator. *Memory* usage for the orchestrator is reduced for all data sets, ranging from 1/6 for 1 sequence to 1/25 for 400 sequences. The reduction in memory usage seems to be close to linear with data size. *CPU* usage shows an even more dramatic improvement—it is not at all affected by data size in the *data-flow delegating* approach, in stark contrast to the *regular* approach, where it increases significantly. The *network traffic* usage is 6 times higher for the orchestrator in the delegating approach for 1 sequence, but is ranging from no change to 1/8 in favor of the delegating approach for 2 to 400 sequences. The improvement in network traffic seems to level out from 100 sequences onwards—with 1/8 reductions for 100, 200, and 400 sequences.

7. CONCLUSION AND FUTURE WORK

We have shown in practice how direct data transfer between standard SOAP web services can be implemented in a centrally orchestrated workflow. The XML schema that we have developed and used to overload service operations, allows the orchestrator to delegate the data-flow to component services, providing the necessary information to the component services for them to exchange the data directly. The changes made to the service interface retain the possibility of accessing the service in a regular manner, so extending a web service with data-flow delegation support does not break existing clients. Moreover, a workflow developer can decide which intermediate data is delegated and which passes through the orchestrator, thus allowing to store data provenance in key steps of the pipeline.

The empirical results confirmed that the data-flow delegation considerably improves the performance of the orchestrator and the entire workflow. The performance gains are higher when the workflow process larger data, similarly to the trend shown by Zhang et al in [24] where data was forwarded as WSRF resources. In contrast to Zhang et al's implementation, our data-flow delegation support is built on top of a standard web service framework. It is a design decision, making our solution framework-independent and applicable to a wider range of services.

Our future plans include adding functionality for handling component service failures during the workflow execution, and support for DAG[14] structured workflows.

Acknowledgement

This work was carried out as part of the eSysbio project (No: 178885) funded by the Research Council of Norway. The authors thank I. Jonassen for comments and suggestions regarding the initial manuscript, and the anonymous reviewers of this conference.

8. REFERENCES

[1] D. Abramson, J. Kommineni, and I. Altintas. Flexible io services in the kepler grid workflow system. In *the International Conference on e-Science and Grid Computing (e-Science)*, Melbourne, Australia, 2005.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standard proposed by BEA Systems, IBM Corporation, and Microsoft Corporation, 2003.

[4] R. Barga and D. Gannon. *Workflows for e-Science*, chapter 2 and 15, pages 9–16. Springer London, 2007.

[5] A. Barker, C. D. Walton, and D. Robertson. Choreographing web services. *IEEE Transactions on Services Computing*, 2(2):152–166, 2009.

[6] A. Barker, J. B. Weissman, and J. I. van Hemert. The circulate architecture: avoiding workflow bottlenecks caused by centralised orchestration. *Cluster Computing*, 12(2):221–235, 2009.

[7] J. Bhagat, E. Nzuobontane F. Tanoh, T Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C. A. Goble. Biocatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research*, 38(2), 2010.

[8] W. Binder, I. Constantinescu, and B. Faltings. Decentralized orchestration of compositeweb services. In *the Proceedings of the International Conference on Web Services (ICWS)*, Los Alamitos, CA, USA, 2006.

[9] The Gene Ontology Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[10] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *the Proceedings of the ACM International Conference on Management of data (SIGMOD)*, Vancouver, Canada, 2008.

[11] D. Habich, S. Preissler, W. Lehner, S. Richly, U. Aßmann, M. Grasselt, and A. Maier. Data-grey-box web services in data-centric environments. In *the International Conference on Web Services (ICWS)*, Utah, USA, 2007.

[12] S. Heinzl, D. Seiler, E. Juhnke, T. Stadelmann, R. Ewerth, M. Grauer, and B. Freisleben. A scalable service oriented architecture for multimedia analysis, synthesis and consumption. *International Journal of Web Grid Services*, 5(3), 2009.

[13] B. Javadi, M. Tomko, and R. O. Sinnott. Decentralized orchestration of data-centric workflows using the object modeling system. In *International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, Washington, DC, USA, 2012.

[14] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua. A reference architecture for scientific workflow management systems and the view soa solution. *IEEE Transactions on Services Computing*, 2(1), 2009.

[15] B. Ludäscher, M. Weske, T. Mcphillips, and S. Bowers. Scientific Workflows: Business as Usual? In *the International Conference on Business Process Management (BPM)*, Ulm, Germany, 2009.

[16] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research Articles. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[17] D. Seiler, S. Heinzl, E. Juhnke, R. Ewerth, M. Grauer, and B. Freisleben. Efficient data transmission in service workflows for distributed video content analysis. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*, Linz, Austria, 2008.

[18] S. Sfakianakis, L. Koumakis, G. Zacharioudakis, and

[14] Directed Acyclic Graph

M. Tsiknakis. Web-Based Authoring and Secure Enactment of Bioinformatics Workflows. In *the Workshops at Grid and Pervasive Computing Conference*, Geneva, Switzerland, 2009.

[19] L. Stein. Creating a bioinformatics nation. *Nature*, 417(6885):119–120, 2002.

[20] S. Subramanian, P. Puntervoll, and P. Sztromwasser. Optimizing the data-traffic of centrally coordinated scientific workflow systems. In *the International Conference on Web Services (ICWS)*, Miami, Florida, USA, 2010.

[21] W. Tan, P. Missier, I. Foster, R. Madduri, D. D. Roure, and C. Goble. A comparison of using taverna and bpel in building scientific workflows: the case of cagrid. *Concurrency and Computation: Practice and Experience (CCPE)*, In-press, Accepted in Oct 2009.

[22] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn. Taverna workflows: Syntax and semantics. *International Conference on e-Science and Grid Computing*, 2007.

[23] W. Yu. Scalable services orchestration with continuation-passing messaging. In *the International Conference on Intensive Applications and Services (INTENSIVE)*, Valencia, Spain, 2009.

[24] D. Zhang, P. Coddington, and A. Wendelborn. Web services workflow with result data forwarding as resources. *Future Generation Computing System*, 27(6), 2011.