Addressing the Shimming Problem in Big Data Scientific Workflows

Aravind Mohan, Shiyong Lu, Alexander Kotov
Wayne State University
Detroit, MI, USA
{aravind.mohan, shiyong, kotov}@wayne.edu

***Abstract*—Substantial amount of research has been done recently to address the shimming problem in scientific workflows, in which a special kind of adaptors, called shims, are inserted between workflow tasks to resolve the data type incompatibility issue. Recently, scientific workflows are increasingly used for big data analysis and processing, which poses additional challenges, such as volume, velocity and variety of data to the shimming problem. One issue is to scale the registration and configuration procedure to a large number of workflow tasks. Another issue is the ease of integrating a large number of remote Web services and other heterogeneous task components that can consume and produce data in various formats and models into a uniform and interoperable workflow. Existing approaches fall short in usability and scalability in addressing these issues. In this paper we 1) propose a new simplified single-component based task model based on extensive experiences and lessons learned from our original multiple-component based task model. The new model separates registration from configuration and eases the process of registering external functional components (such as Web services) into p-workflows; 2) propose a shim generation algorithm that elegantly solves the shimming problem raised by Web service based scientific workflows; and 3) we integrate MongoDB , a NoSQL document-oriented database system for storing and managing large-scale unstructured documents. A new version of the DATAVIEW system has been developed to support the proposed techniques and a case study has been conducted to show the feasibility and usability of our proposed techniques.**

Keywords-shimming; scientific workflow; big data;

I. INTRODUCTION

In recent years, the term "big data" has become a buzzword in both industry and academia. It is frequently used to refer to the method of processing and analyzing large amounts of heterogeneous types of data to mine hidden patterns, correlations, trends, and other useful information to guide enterprise decisions and strategies for cutting cost or increasing revenues [14, 15]. Scientific workflows, which were originally developed for automating the data analysis process of scientific data (often large in volume and vary in types) to enable and accelerate scientific discovery, are now increasingly used for the processing and analysis of business data as well. However, the challenges of big data, including volume, velocity, and variety pose additional challenges to scientific workflow research.

The shimming problem has received much attention in the scientific workflow community [6, 11, 12, 13], in which special kinds of adaptors, called *shims*, are inserted between workflow tasks to resolve the data type incompatibility issue. However, to meet the challenges of big data, several issues in the existing approaches need to be examined. The first issue is to scale the registration and configuration procedure to a large number of workflow tasks, which is critical for big data workflows. The second issue is to ease the integration of a large number of remote Web services and other heterogeneous task components, such as command line executables, that can consume and produce data in various formats and data models into a uniform and interoperable workflow. In particular, we examine automatic shim generation techniques for third-party Web services, whose development were not workflow-aware.

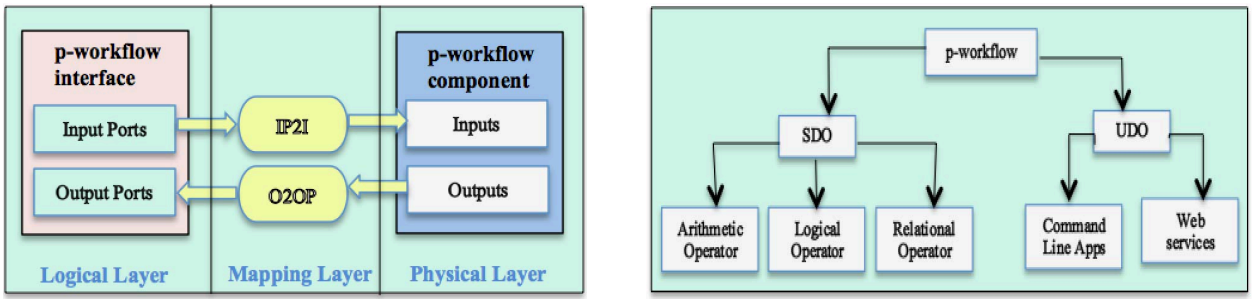
The contributions of the paper are: 1) based on extensive experiences and lessons from our previously proposed multiple-component based task model [12], we propose a new simplified single-component based task model. The new model separates registration from configuration and eases the process of registering external functional components (such as Web services) into p-workflows; 2) we propose a shim generation algorithm, called *GenShims*, that elegantly solves the shimming problem raised by Web service based scientific workflows; 3) we integrate MongoDB , a NoSQL document-oriented database system for storing and managing large-scale unstructured documents. A new version of the DATAVIEW system has been developed to support the proposed approaches.

II. PRIMITIVE WORKFLOW MODEL

Primitive workflows (a.k.a. tasks or p-workflows) are the basic building blocks of a scientific workflow. Composite workflows were used to represent workflows that consist of multiple workflows. Our previously proposed task model [12], contains multiple components inside a single task and hence complicates the registration process as complex shimming needs to be done between the components at design time in order to register any task as a p-workflow. In our new model, we elegantly solve this problem by having only one component inside a task and ease the registration process. At design time, we also provide a configuration mechanism through which we get the input and output port information from the user, when necessary.

One appealing feature of our new primitive workflow model is that it provides an abstraction technique, in which different heterogeneous task components can be abstracted into uniform p-workflows so that they can interoperate with one another. While a task manager needs to be aware of the

978-1-4799-5066-9/14 $31.00 © 2014 IEEE
DOI 10.1109/SCC.2014.53



(a) (b)

Figure 1 (a) our proposed primitive workflow model; and (b) a classification of p-workflows.

implementation details of each p-workflow in order to know how to register it as a p-workflow; once registered, such implementation details are hidden from the workflow engineer, who can drag and drop any p-workflow into the design panel, regardless of how it is implemented and connect them with one another into a composite workflow.

In our system, we provide a user friendly GUI for the design and modification of p-workflows. As shown in Figure 1(b), using our model, we classify p-workflows into the following:

1) *System Defined operations.* They are also called built-in p-workflows. These p-workflows come with a DATAVIEW installation. Examples of such p-workflows include the logical operations (such as AND, OR, and NOT) and relational algebra operators (such as SELECTION, PROJECTION, and UNION).
2) *User Defined Operations.* These are the p-workflows that are provided by task engineers through registration and configuration. The task engineer provides the task interface and its implementation detail during the registration process. Examples of such p-workflows include command line based applications and web service based applications.

In order to simplify the workflow design process and to emphasize the usability of our system, we maintain a workflow repository through which our users can easily access existing workflows and use them to compose more complex workflows in the workflow design panel Furthermore, in our model, we separate workflows into two categories: reusable workflows and executable workflows. Reusable workflows are complex operations designed without connecting any input and output data products to the workflow, thereby they are mainly used by the workflow engineer to build reusable templates. On the other hand, executable workflows contain input and output data products connected to the input ports of workflows, and they are mainly used by data scientists to perform data analysis and processing. Workflow design is supported through our workflow specification language known as SWL that defines a scientific workflow according to the DATAVIEW Workflow Model [7], through which one can create scientific workflows with different layers of granularity in a nested manner. SWL, which is automatically created on the server side while designing any workflow, is formulated by translating the workflow diagram that is represented in our workflow visualization language (mxGraph). mxGraph is mainly used at the client side to render and visualize a workflow as a diagram. While a workflow is saved in our system, both the mxGraph and SWL specifications are serialized and stored into our database. As shown in Figure 1(a), our proposed new primitive workflow model consists of the following three layers:

1) **Logical Layer**: The Logical layer contains the p-workflow interface that models the port details of the workflow such as input port ID, input port type, output port ID, and output port type. At run time, the data flow occurs through the data channel connecting the source workflow constituent to the destination workflow constituent.
2) **Mapping Layer**: The Mapping layer contains a list of mapping information such as mapping of an input port of the p-workflow interface to an input of the p-workflow component (IP2P) and mapping of an output of the p-workflow component to an output port of the p-workflow interface (O2OP).
3) **Physical Layer**: The Physical layer contains only one p-workflow component at a time, that models the service and or application that is used to implement the task. Hence registration of p-workflow is made simple. Heterogeneous characteristics of the p-workflow are captured and modeled in this layer including the inputs, outputs, location of workflow component (such as Web service WSDL file, command line executable file.)

Using our primitive workflow model, task engineers can easily register a command line based application as a p-workflow by uploading their executable into our DATAVIEW server and providing configuration details, such as input type, input mode, output type, and output mode. As shown in Tables 1(a) and 1(b), we formulated a mapping table based on different cases that a command line application can fall under in respect to the input and output types. We identified the following 6 input modes: 1) ByValue – the IP2I mapper pass the value from the input port as an input argument during component invocation; 2) ByFile – the IP2I mapper generate unique file name, create the file and write content from the input port into the file, pass file name as an argument during component invocation; 3) ByFixedFile – the IP2I mapper create a file with the fixed

Table 1(a) IP2I mapping table.

Input Port name	Input Port type	Cmd Line input	Input Mode
ip1	Integer	arg1	ByValue
ip2	Integer	arg2	ByFile
ip3	Integer	null	ByFixedFile
ip4	Integer	stdin	ByStdin
ip5	Integer	environment variable	Byenv
	null	arg3	ByConst

Table 1(b) O2OP mapping table.

Cmd Line output	Output Mode	Output Port name	Output Port type
stdout	ByValue	op1	Integer
null	ByFixedFile	op2	Integer
exit code	ByValue	op3	Integer
arg4	ByFile	op4	Integer
environment variable	ByEnv	op5	Integer
arg5	ByFile	null	null

file name and write content from the input port into the file, pass file name as an argument during component invocation; 4) ByStdin – the IP2I mapper gets the value from the input port and pipe it to the input stream during component invocation; 5) ByEnv – the IP2I mapper will create environment variable with a user driven name and take the value from the input port and write into it during the component invocation; 6) ByConst – the IP2I mapper simply add a constant value as one of the argument during component invocation. On the other hand, we identified the following 3 output modes: 1) ByValue –the O2OP mapper simply takes the output that is returned during the component execution which could be either stdout or exit code and binds it to the output port; 2) ByEnv – the O2OP mapper will fetch the value from the environment variable using the name provided by user during registration and bind the value to the output port during component execution; 3) ByFile – the O2OP mapper will simply fetch the value from the file name which is provided through one of the arguments and bind the value to the output port during component execution.

In order to demonstrate the strength of our model, we implemented and tested six cases in which a command line executable can be registered and configured into a p-workflow in our DATAVIEW system. Due to space limitation, we have shown only one case:

Case 1: suppose we created a command line java application *addition1.class* which takes two command line arguments, both of Integer type, and produces the sum of the two integers as the stdout. After converting *addition1.class* into a p-workflow *pw_addition1*, the IP2I mapper will take the two integers from input ports i1, i2 and then use them as two command line arguments (inputs) for pw_addition1, after the execution of pw_addition1, the O2OP mapper will route the output in the stdout to output port o1 for pw_addition1. Figure 2 shows a simple example of a scientific workflow of case1 implemented in our system.

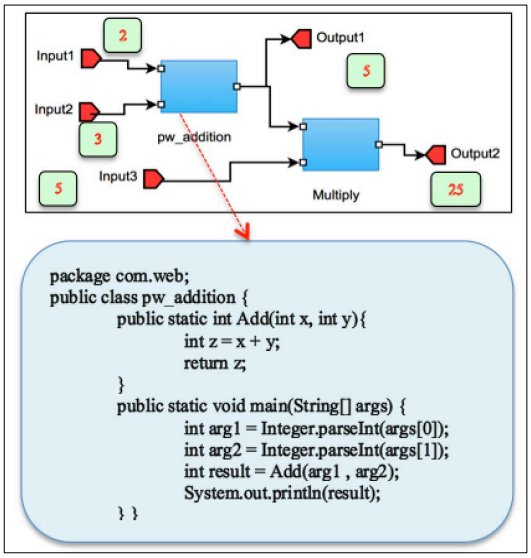


Figure 2 Command line based p-workflow.

We identify the following advantages of our primitive workflow model:

- **Domain Proficiency**: Since the developers of these third party applications are experts in the domain and the software components are well studied and implemented, using it within a scientific workflow is a big advantage and simplifies the workflow design process through the anatomy of reusability and sharing.
- **Reliability**: These third party applications are highly reliable because they are used by many scientists all over the world and hence most of the problems/bugs in the software will be fixed promptly.
- **Infrastructure**: Most of third party applications such as Web services are hosted in a remote server and hence high computation cost is not an issue as the infrastructure is available free through invoking the Web service.
- **Support**: Many third party applications are free to download, install and also have a forum where people discuss various issues and fixes about these software components.

III. Shim Generation Algorithm for Web Service based Scientific Workflows

In this section, we first propose an approach to solve the shimming problem in our DATAVIEW system, that occurs during registration of any Web service based p-workflow through a user-driven configuration mechanism and then provide a shim generation algorithm to translate a p-workflow into a composite workflow by wrapping two special adaptors known as p*Combiner* and p*Splitter* shims around the p-workflow. Next, we demonstrate an example of how our *GenShims* algorithm works.

3.1 Addressing the shimming problem

Because of the heterogeneous nature of scientific workflows, our system provides a platform to register

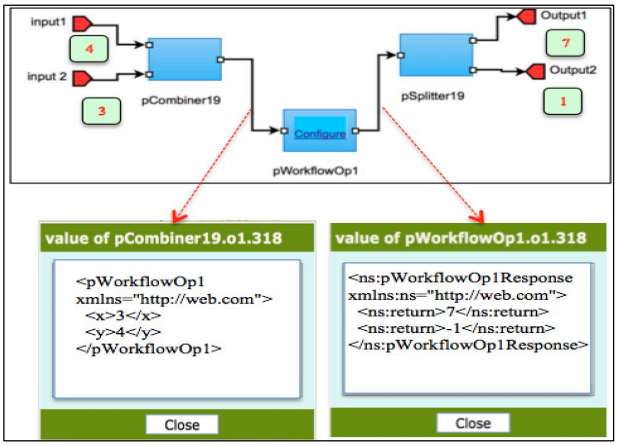


Figure 3(a) Web service based p-workflow.

primitive-workflows from different sources such as Web services, command line applications. However, this then becomes an issue because of the incompatibility of data types between the input/output port of a p-workflow and input/output of component connected to it. Due to the shimming problem that exists during the workflow composition, at run time the p-workflow is not interoperable with other types of workflow constituents. Through a user driven configuration mechanism, we separate the shimming problem from registration and thereby making p-workflow registration simple and usable. After registering the p-workflow, the shimming problem is solved by configuring the input and output port of any arbitrary p-workflow and the system automatically creates instances of two special kinds of shims known as *pCombiner* and *pSplitter*. Finally a new wrapper workflow is created on the fly during the p-workflow configuration with the instances of *pCombiner* and *pSplitter* inserted into the new wrapper workflow. In this way we hide the shim at the workflow level and abstract the existence of shim from the data scientist perspective, but during workflow execution, a shim automatically does the conversion based on the configuration details provided by the workflow designer. Data scientists would simply need to drag and drop the wrapper workflow, connect data products to it and execute the workflow to view the results.

In our DATAVIEW system, a task engineer can easily register any arbitrary Web service operation by providing the system with the WSDL file of any arbitrary Web service. We implemented WSDL parser functionality within our system that can automatically get the input and output of an operation from the WSDL and convert it into the input and output port of the workflow. Additionally we generated automatically two special shims: *pCombiner* and *pSplitter*. *pCombiner* shim, is a special kind of shim adaptor that takes in any number of arbitrary data from the input ports of arbitrary types and create a new data product of type XML and bind the XML data to the output port. The *pSplitter* shim, is another special kind of shim adaptor that

```
<workflowSpec>
<workflow name="ws_pWorkflowOp1" root="true">
<workflowInterface>
 <inputPorts>
<inputPort>
<portID>i0</portID>
<portName>i0</portName>
<portType>Integer</portType>
</inputPort>
<inputPort>
<portID>i1</portID>
<portName>i1</portName>
<portType>Integer</portType>
</inputPort>
</inputPorts>
 <outputPorts>
<outputPort>
<portID>o0</portID>
<portName>o0</portName>
<portType>Integer</portType>
</outputPort>
<outputPort>
<portID>o1</portID>
<portName>o1</portName>
<portType>Integer</portType>
</outputPort>
</outputPorts>
</workflowInterface>
<workflowBody mode="graph-based">
<workflowGraph>
<workflowInstances>
<workflowInstance id="pWorkflowOp1">
<workflow>pWorkflowOp1</workflow>
</workflowInstance>
<workflowInstance id="pCombiner19">
<workflow>pCombiner19</workflow>
</workflowInstance>
<workflowInstance id="pSplitter19">
<workflow>pSplitter19</workflow>
</workflowInstance>
</workflowInstances>
<dataChannels>
<dataChannel from="pWorkflowOp1.o1" to="pSplitter19.i1"/>
<dataChannel from="pCombiner19.o1" to="pWorkflowOp1.i1"/>
</dataChannels>
</workflowGraph>
<G2W>
<inputMapping from="this.i0" to="pCombiner19.i1"/>
<inputMapping from="this.i1" to="pCombiner19.i2"/>
<outputMapping from="pSplitter19.o1" to="this.o0"/>
<outputMapping from="pSplitter19.o2" to="this.o1"/>
</G2W>
</workflowBody>
</workflow>
</workflowSpec>
```

Figure 3(b) Workflow Specification (SWL) with shim inclusion.

takes in data from the input port of type XML and extract the elements inside it in a unique manner and split the result into multiple outputs, which are then bound to multiple output ports. Input port details of the *pCombiner* such as (no of input ports, input port types) and output port details of the *pSplitter* such as (no of output ports, output port types) are identified during the p-workflow configuration. As shown in Figure 4, the proposed *GenShims* algorithm takes an input p-workflow *ws1* and converts it into a composite workflow *c-ws1*.

In order to create the composite workflow *c-ws1*, our algorithm creates both workflow specification (SWL) and visualization (mxGraph) for the workflow automatically. Our *pCombiner* is IP2I mapper for Web service based p-workflow and each time *GenShims* algorithm is executed, it creates a new instance of pCombiner. On the other hand, the *pSplitter* shim is an O2OP mapper for Web service based p-workflow and each time the *GenShims* algorithm is executed, it creates a new instance of *pSplitter*. Figure 3(b), shows the SWL of ws*_pWorkflowOp1* that contains the *pCombiner*, *pSplitter* and *pWorkflowOp1* (p-workflow) respectively in our DATAVIEW system. Our approach currently works for Web services with primitive types of int and string for the following cases of web service operation: one input and one output, one input and multiple outputs, multiple inputs and one output, multiple inputs and multiple outputs, no input and one output, no input and multiple outputs, no input and no output, one input and no output, multiple inputs and no output. In the future, we will explore the above cases with complex data types as inputs and outputs.

3.2 Example of Shim Generation.

Figure 3(a) shows an example of shim generation for Web service based workflows. To simply the explanation, we developed a Web service *ws_addsubtract* in Java, which takes a pair of integers *a* and *b*, and returns their sum and difference, respectively as two integers *c* and *d*. After registering the WSDL URL of this Web service. (http://dmsg2.cs.wayne.edu/axis2/services/pWorkflow1?wsdl) in DATAVIEW, the system automatically extracts all the operations of the Web service. In our case, the user selects the only operation available, *pWorkflowOp1*, and the system automatically convert it into p-workflow *pWorkflowOp1*. This workflow has one input port of XML type and one output port of XML type. The user can then click the *configure* link in this workflow, which allows the user to choose the number of input ports (which is 2) and the number of output ports (which is also 2). After the completion of the configuration, two shims, *pCombiner19* and *pSplitter19*, are generated automatically, and a new complex workflow *cw_pWorkflowOp1* is constructed. Workflow cw_pWorkflowOp1 contains two input ports *a* and *b*, and two output ports *c* and *d*. Figure 2 shows one case of execution with inputs a=4 and b=3, displaying all the intermediate values of each step.

IV. INTEGRATION OF NOSQL DATABASE

NoSQL databases are distributed and schemaless databases that have recently emerged as a technology developed to address the need to store and process huge volumes of data. Existing NoSQL databases such as Riak, MongoDB , Cassandra, HBase, Neo4j enable horizontal scalability across a large number of commodity servers. In DATAVIEW, we integrated MongoDB within our

Algorithm: GenShims
Input: Primitive Workflow ws1
Output: Composite Workflow c_ws1
Begin

1. **If** ws1 is of type web service
2. **Then** extract configuration details such as input port name{In), input port type(It), output port name(On), output port type(Ot) by parsing the wsdl file of the primitive workflow ws1.
3. Auto populate port details in the configuration section of the primitive workflow ws1, with an option for the users to modify or add new port information.
4. Create a new instance of the combiner shim (C) , which takes in arbitrary number of input ports of name In and type It and combine them into an output Oc of type XML.
5. Create a new instance of the splitter shim (S) , which takes in XML input Is and parse the XML to extract all the values inside it and split them into a series of outputs of name On and type Ot.
6. Create SWL for composite workflow c-ws1 with the workflow interface that has input ports In with port type It and output ports On with port type Ot.
7. Append the workflow instances section to SWL that was created in step 6 with the instances pCombiner, ws1,and pSplitter.
8. Append the dataChannel section to SWL with the following source -> destination data flow construct.
    a) Output from the combiner shim (Oc) to the input of the primitive workflow ws1.
    b) Output of the primitive workflow ws1 to the input of the splitter shim (Is).
9. Create the mxGraph diagram for the composite workflow c_ws1 with the input port In of type It connected to pCombiner input port. Output of pCombiner connected to the input of ws1. Output of ws1 connected to the input of pSplitter. Output of pSplitter connected to the output port On of type Ot
10. **Else**
11. No shim required to be inserted.

End Algorithm

Figure 4 GenShims Algorithm.

data product manager to store and manage different data products. MongoDB is a scalable, open source document-oriented database that is commonly classified as a NoSQL database. MongoDB uses JSON-like documents with no schema to store data inside different collections. In DATAVIEW, execution of a scientific workflow consumes and produces huge amount of distributed data objects. These data objects are heterogeneous of various data types. A scientific workflow might use mixed data types as the data could be any form and aligns well with the schemaless and scalable nature of MongoDB . In DATAVIEW, we provide an abstraction of data objects stored in MongoDB as a data product.

Three challenges of Big Data namely the volume, velocity and variety is a key issue focused today in both academia and industry. Data can be from different sources and the system should have the capability to some how

register and import the data and perform analysis and process the data. One of the main advantages of integrating MongoDB in our system is that, it is horizontabally scalable across the commodity servers and hence the data from the scientists can be shipped through our data product manager with high performance. Another advantage is MongoDB offers MapReduce support at the database level to process huge amounts of data into meaningful aggregated results. MongoDB applies the map phase to each document inside the collection that matches the query condition and emits a (key, value) pairs. And then the reduce phase is run for those keys that contain multiple values and data is collected and aggregated to form a new collection in the database. Our solution to address the challenge occurred by such a large variety of types in big data is to use a custom approach to classify different data types and then perform shimming to transform data into one of the DATAVIEW data types. As shown in Table 2, we defined the notion of DATAVIEW data type in our system and broadly classify data products into the following categories: 1) scalar 2) relational 3) collectional 4) XML and 5) file.

V. CASE STUDY

In our DATAVIEW system, we created several automotive consumer review big data primitive workflows in order to collect, merge, extract, and analyze the raw data from automotive review websites, including KBB, Edmunds, and MSN Autos. As part of our case study, we deployed a MongoDB sharded cluster in FutureGrid that provides a schemaless, horizontabally scalable, and MapReduce enabled data storage and processing in a cluster enivronment. In order to deploy the MongoDB cluster, we created one VM instance with 4 GB memory and 40 GB disk space. In addition, we created 3 VM instances with each 2 GB memory and 40 GB disk space and setup the config servers which contains the metadata information such as shard and block level information. In MongoDB, the config servers are single point of failure and hence we created 3 instances for replication purpose on node failure. Finally we created 3 more VM instances with each 4 GB memory and 60 GB disk space and setup the mongod shards that contains the actual data sharded (split inside chunk and new chunks). In addition to deploying the sharded cluster, we also integrated the MongoDB Java driver within our DATAVIEW system, in order to make a connection with the MongoDB cluster and perform several operations on the data.

As shown in Fgiure 5(a), we created an automotive consumer review analysis workflow. Firstly, we created a non map reduce p-workflow known as *SearchNonMR* that takes in three inputs (AutoReview, FieldName1, SearchTerm1) and query the *FieldName* on the master node of the MogoDB cluster with an or condition to find the match for the list of search terms provided by the user such as "MPG, Fuel Economy, Excessive comsumption". During our experiments we found that the execution of the workflow is time consuming and takes around 629 seconds for inserting 8425 (11.25 MB) records into a new collection in the MongoDB cluster. Secondly, we created two map reduce p-workflows known as *SearchLevel1MR* and *SearchLevel2MR* to process the consumer review dataset in a map reduce manner and break down the data into State and City level. *SearchLevel1MR* takes in 3 inputs (AutoReview, FieldName2, SearchTerm2) and execute the map reduce program written with a mapper that emits (Search Term, Review) based on a REGEX match condition and a reducer that aggregates all the review for distinct search terms. Output of the map reduce function is automatically binded to a new collection inside the MongoDB cluster. Output from the *SearchLevel1MR* is passed as input to *SearchLevel2MR* which would execute a map reduce program with a mapper that loops through each item in the aggregated review and for each review item it emits for example (State Name, Review) based on a REGEX match condition and a reducer aggregates all the reviews for distinct states. During our experiments we found that the execution of the workflow takes around 17 seconds for creating an output collection of size 6 MB. Thirdly, we created a workflow to get the list of users who posted reviews in a list of states. During workflow execution a map reduce program is executed to get the list of reviews that contain the match for one or more of the items in the list of states and the aggregated review are stored into a new collection in MongoDB. Fourthly, we created a workflow to break down the review into Year, Month and Date and hence the data scientist can easily visualize how many reviews were posted for a particular year and then perform more fine grained analysis such as get the reviews posted for a particular month and date, etc.

In order to perform data collection, as shown in Figure 5(b) we created a series of command line based p-workflows known as KBB, Edmunds and MSN where data crawling procedures are executed internally and passed into a customized HTML parser, to extract consumer review information. Output from the three data collection workflows are passed as input to the data integration

Table 2 Data Types supported in DATAVIEW.

DATAVIEW Data Type	Description
Scalar	to store data of following types: string, decimal, integer, non-positive integer, non-negative integer, positive integer, negative integer, unsigned int, unsigned long, unsigned short, unsigned byte, double, float, long, int, short, byte, boolean.
Relational	to store relation that are represented as tables.
Collectional	to store collection that are represented as key-value pairs.
XML	to store xml documents that conforms to well formed xml value
File	to upload the file on server side and store physical path of the file.

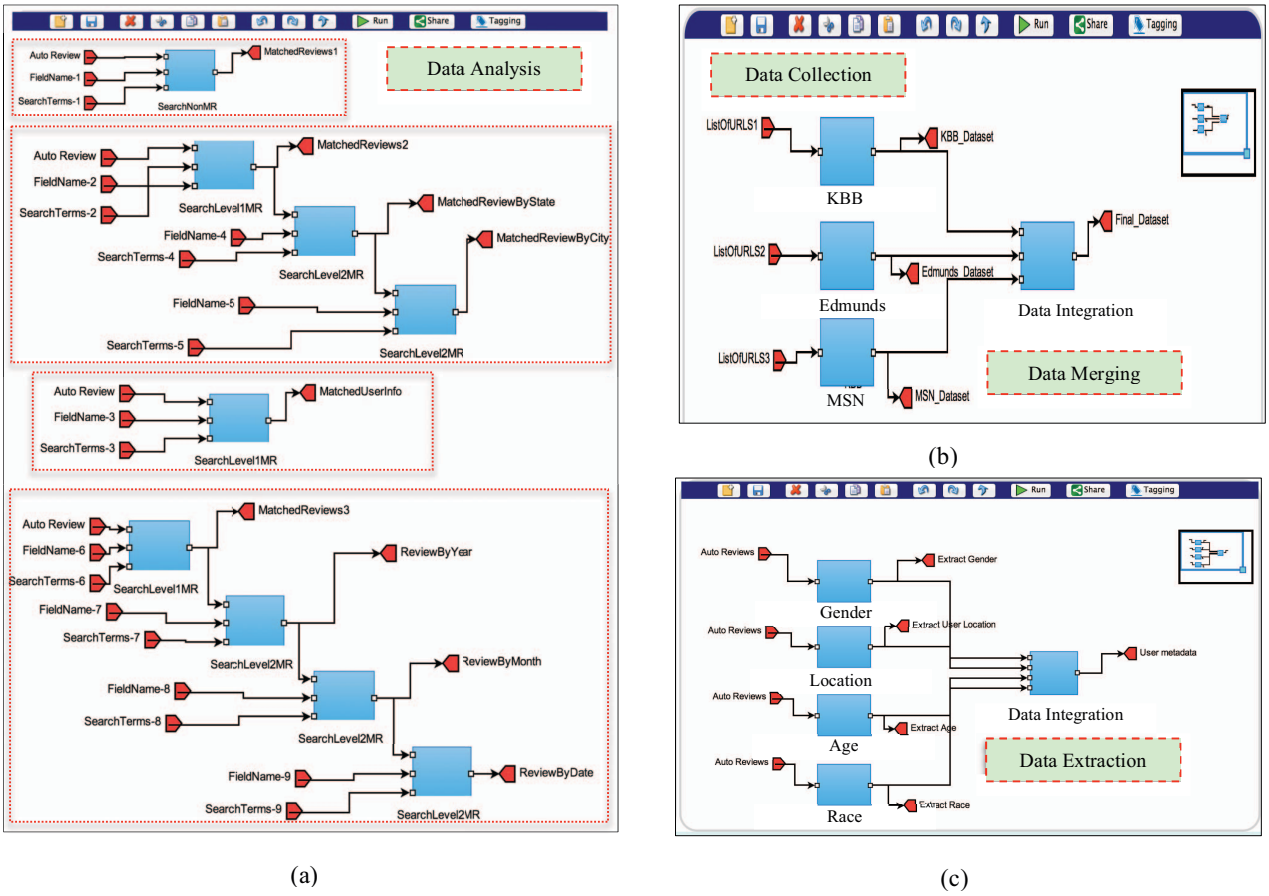


Figure 5 (a) Automotive Consumer Review Analysis Big Data Workflow; (b) Data Collection/Integration Workflow; (c) User Meta Data Extraction Workflow.

workflow where the data is integrated together as one collection by executing the union operation on the input dataset.

As shown in Figure 5(c), we created data extraction p-workflows to extract the users meta data information automatically from the consumer reviews. Towards this end, we created a command line based p-workflow known as *Gender* which is primarily used to extract the gender of the user who posted the review. Suppose there is a review that contains the following text "My wife loves Ford Focus 2014", the existence of the word "My wife" tells that the gender of the user is Male and hence a new output collection is created which contains the user meta data (Gender). In addition to the Gender, we created three more p-workflows known as *Location*, *Age* and *Race* to extract the User meta data information based on the existence of predefined set of words and pass the output data from the user meta data workflows to the data integration workflow where the data is integrated together and stored as a new collection. in the MongoDB cluster. Due to space limitation, in this paper, we have not shown much of the implementation details of the user meta data extraction workflows.

VI. Related Work

The notion of the shimming problem was first coined in [6]. Due to the heterogeneous nature of the data involved in solving complex scientific problems, output of one program or application will not be always compatible with another program or application and hence the interoperability of both programs and applications is a challenge. The challenge of interoperability between the mixed data types is often termed as "shim" and there has been much work done in classifying and solving the shimming problem. Shim is broadly classified into 1) semantic compatibility 2) syntactic compatibility. Hull et al. [6] introduced shims as a technique to align or mediate mismatching third party Web services that has incompatible input and output. Other work has been done to solve the shimming problem in literature such as Szomszor et al. [11] in which a mapping language is proposed to use OWL ontologies in order to capture the semantics of a data source and the instances of these ontology concepts are used to support conversion of data between different formats. Although this technique introduces the ontological model that can be used to solve the shimming problem to some extent from the semantic

incompatibility perspective, it is very complex to implement and is not usable. Ambite and Kapoor [1] proposed an approach to solve the shimming problem by automatically inserting the shims into scientific workflow but the approach only supports relational data and does not have any contribution towards other data types.

Lin et al. [12] proposed the task model in scientific workflow and classified the shimming problem into TYPE-I and TYPE-II shimming problems. Further this approach uniquely allows shims to be either invisible or visible at the workflow level, supporting both functional and operational perspective of scientific workflows Although the work was well cited and accepted in the scientific workflow community, the proposed task registration model is much complicated as the model contains more than one component within a single task and the shimming needs to be done while registering a task into the system through complex mapping between the components within a task. Kashlev et al. [13] proposed to reduce the shimming problem in scientific workflow to the runtime coercion problem and defined the notion of well typed workflow. But this approach did not address the data mapping challenges incurred during registration of a p-workflow and did not provide any solution to guarantee interoperability between the p-workflow and other upstream/downstream workflows connected to it .

None of the above technique addresses the shimming problem that exists in registering third party applications such as web services and command line based applications as a p-workflow , which is the focus of this paper. In this paper, we provide two solutions:

1) For a command line based application, we formulated the IP2I and O2OP mapping table and proposed the solution in a case by case approach.
2) For a Web service based application, we provided a solution to separate the registration from the configuration, thereby easing the registration process of a Web service operation as a p-workflow into our DATAVIEW system. Shimming is automatically done inside the p-workflow using the user driven configuration mechanism. In addition, our approach also parses the WSDL file automatically to get the input and output information and hence automates the conversion of a workflow unaware Web service operation as a p-workflow from end to end and manage to guarantee interoperability with both upstream and downstream workflows.

In this paper, we also addressed challenges incurred during analyzing and processing large datasets in scientific workflow, by integrating MongoDB, a NoSQL database within our DATAVIEW system as a preliminary step to perform big data analysis and processing.

VII. Conclusions and Future Work

In this paper, we first proposed a new primitive workflow model, to overcome the limitations of the previous multiple-component based task model, in which we separate the registration from the configuration of p-workflow, thereby making the registration process simple and usable. Our new model eases the registration process and hence there is no need to do mapping between multiple task components inside the p-workflow. Second, we proposed a shim generation algorithm to solve the shimming problem raised in Web services by automatically inserting invisible shims and wrapping it around the p-workflow. Third, we integrated MongoDB , an open source document oriented database, into our DATAVIEW system in order to support big data management and processing. Finally, we implemented the proposed models and algorithm in our DATAVIEW system and presented a case study to validate them. Ongoing work includes extension of the primitive workflow model to support registration and execution of a p-workflow in cloud and grid computing environments.

Acknowledgement

This work is partially supported by U.S. National Science Foundation under CCF 0845711.

References

[1] J. Ambite and D. Kapoor. Automatically composing data workflows with relational descriptions and shim services. In ISWC/ASWC, pages 15–29, 2007.

[2] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. VisTrails: visualization meets data management. In SIGMOD Conference, pages 745–747, 2006.

[3] S. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In SIGMOD Conference, pages 1345–1350, 2008.K. Elissa, “Title of paper if known,” unpublished.

[4] E. Deelman and A. Chervenak. Data management challenges of data-intensive scientific workflows. In CCGRID, pages 687–692, 2008.

[5] D. Hull, R. Stevens, and P. Lord. Describing web services for user-oriented retrieval. In Proc. of W3C Workshop on Frameworks for Semantics in Web Services, pages 9–10, 2005.

[6] D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. In AKT- SWS04, 2004.

[7] C.Lin,S.Lu,X.Fei,A.Chebotko,Z.Lai,D.Pai,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):79–92, 2009.

[8] C.Lin,S.Lu,Z.Lai,A.Chebotko,X.Fei,J.Hua,andF.Fotouhi. Service-oriented architecture for VIEW: A visual scientific workflow management system. In IEEE SCC, pages 335–342, 2008.

[9] B.Ludascher,S.Bowers,T.McPhillips,andN.Podhorszki. Scientific workflows: More e-science mileage from cyberinfrastructure. In e-Science, pages 145–152, 2006.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17):3045– 3054, 2004.

[11] M. Szomszor, T. Payne, and L. Moreau. Automated syntactic medation for web service integration. In ICWS, pages 127– 136, 2006.

[12] C. Lin, S. Lu, X. Fei, D. Pai and J. Hua, "A Task Abstraction and Mapping Approach to the Shimming Problem in Scientific Workflows", *IEEE International Conference on Services Computing (**SCC**)*, pp.284-291, Bangalore, India, 2009.

[13] A. Kashlev, S. Lu, A. Chebotko "Coercion Approach to the Shimming Problem in Scientific Workflows", *IEEE International Conference on Services Computing (**SCC**)*, pp.416-423, Santa Clara, CA, 2014.

[14] S. Singh, N. Singh, "Big Data Analytics", *IEEE International Conference on Communication, Information and Computing Technology (**ICCICT**)*, pp.1-4, Mumbai, 2012.

[15] http://www.01.ibm.com/software/data/infosphere/hadoop/what-is-big-data-analytics.html.