Design, Implementation, and Evaluation of a Tight Integration of Database and Workflow Engines

Peter Reimann, Holger Schwarz, and Bernhard Mitschang

Institute of Parallel and Distributed Systems, University of Stuttgart, Germany
{Firstname.Lastname}@ipvs.uni-stuttgart.de

Abstract. Accessing and processing huge amounts of heterogeneous and distributed data are some of the major challenges of data-intensive workflows. Traditionally, the descriptions of such workflows focus on their data flow. Nevertheless, control-flow-oriented workflow languages are increasingly adapted to the needs of data-intensive workflows. This provides a common level of abstraction for both data-intensive workflows and classical orchestration workflows, e. g., business workflows, which then enables a comprehensive optimization across all workflows. However, the problem still remains that workflows described in control-flow-oriented languages tend to be less efficient for data-intensive processes compared to specialized data-flow-oriented approaches. In this article, we propose a new kind of optimization targeted at data-intensive workflows that are described in control-flow-oriented languages. We show how to improve efficiency of such workflows by introducing various techniques that partition the local data processing tasks to be performed during workflow execution in an improved way. These data processing tasks are either assigned to the workflow engine or to the tightly integrated local database engine. We evaluate the effectiveness of these techniques via various test scenarios.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures; H.2.8 [**Information Systems**]: Database Applications; H.4.1 [**Information Systems Applications**]: Office Automation

Keywords: Data-Intensive Workflow; Improved Local Data Processing; Scientific Workflow; Simulation Workflow

1. INTRODUCTION

Workflows have long been used to meet the needs of IT support for business processes. They are compositions of tasks by means of causal or data dependencies that are carried out on a computer using a workflow management system (WfMS) [Leymann and Roller 1999]. Recently, the workflow technology has found application in the area of scientific computing and simulations for implementing complex scientific applications and the term *scientific workflow* has been coined [Taylor et al. 2007]. Many of these scientific workflows treat data and their processing as first-class citizens. Such data-intensive workflows typically process huge amounts of possibly distributed and heterogeneous data and carry out a multiplicity of complex data processing steps that may be directly reflected in the workflow definitions [Deelman and Chervenak 2008]. Examples are workflows for analyzing previously generated data, for reducing models of a simulation problem, and for protein modeling and analysis.

Traditionally, the descriptions of data-intensive workflows focus on their data flow instead of the control flow due to several benefits of this kind of workflow descriptions. This particularly includes optimization opportunities for massive data processing [Ludäscher et al. 2009], [Zinn et al. 2010], [Coutinho et al. 2010]. Nevertheless, control-flow-oriented workflow languages are increasingly adapted to the needs of data-intensive workflows since they have some advantages for modeling and executing such workflows as well [Böhm et al. 2007], [Slominski 2007]. They provide a common level

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1), Florian Wagner for implementing ODE-TI, and the anonymous reviewers of the journal submission process for their valuable remarks and suggestions.

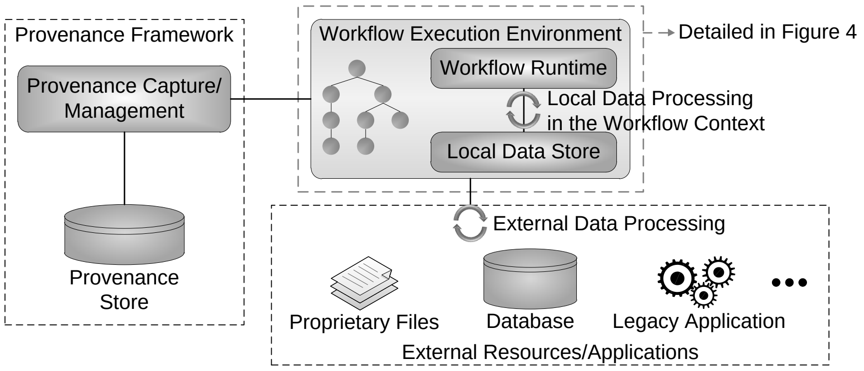

Fig. 1. Data processing during workflow execution phase.

of abstraction for both data-intensive workflows and classical orchestration workflows such as business workflows. This enables a comprehensive re-engineering or optimization across all kinds of workflows [Radeschütz and Mitschang 2009], [Vrhovnik et al. 2007], [Böhm et al. 2007]. Furthermore, it results in a generic solution, i. e., being applicable in multiple domains of scientific or business applications and also for workflows combining multiple domains [Reimann et al. 2011].

The life cycle of a scientific workflow typically consists of the phases workflow design, workflow preparation or deployment, workflow execution, and post-execution analysis where result data, provenance data, and other metadata are archived and analyzed [Ludäscher et al. 2009], [Deelman and Chervenak 2008]. Figure 1 presents a high-level view of a workflow processing architecture including a workflow execution environment, external resources and applications, and a provenance framework to capture, manage, and store provenance data [Freire et al. 2008]. In this article, we focus on the workflow execution phase, in particular on the processing of application data during this phase. For some data sets, it may be appropriate or even inevitable to process them directly in the workflow execution environment. The other option, which is particularly interesting for larger data sets, is to outsource data processing to external resources or applications. Here, many approaches exist that deal with access mechanisms for proprietary data sources or legacy applications [Reimann et al. 2011], [Görlach et al. 2011] or focus on specific optimization opportunities, e. g., late binding of resources or re-engineering of workflow models [Vrhovnik et al. 2007]. However, optimization opportunities for processing high amounts of data directly in the workflow context, in particular for control-flow-oriented workflow languages, have largely been neglected in previous work.

In this article, we focus exactly on this setting and show how to extend the system architecture to transparently improve the local data processing in a workflow execution environment. We introduce various techniques to partition the local data processing tasks in an improved way. These data processing tasks are either assigned to the workflow runtime or to a tightly integrated local database engine thus exploiting its data processing capabilities. We evaluate the effectiveness of these techniques by means of various test scenarios and come up with possible indicators when to use either the workflow runtime or the local database engine. Altogether, this offers a great potential for improved performance and reliability of local data processing in data-intensive workflows and it broadens the set of such workflows that may be described in terms of control flow. It forms the desired generic solution for all kinds of workflows in multiple domains. Furthermore, it increases the extensibility of current workflow execution environments by additional data management functionalities, e. g., the usage of a geographic information system may support operations tailor-made to spatial problems.

The rest of this article is organized as follows: Section 2 deals with the state of the art of data processing in workflows. In Section 3, we introduce our approach and discuss its benefits in Section 4. Afterwards, Section 5 shows the results of our evaluation. We highlight major related work in Section 6, and Section 7 concludes and gives a brief outlook for future research.

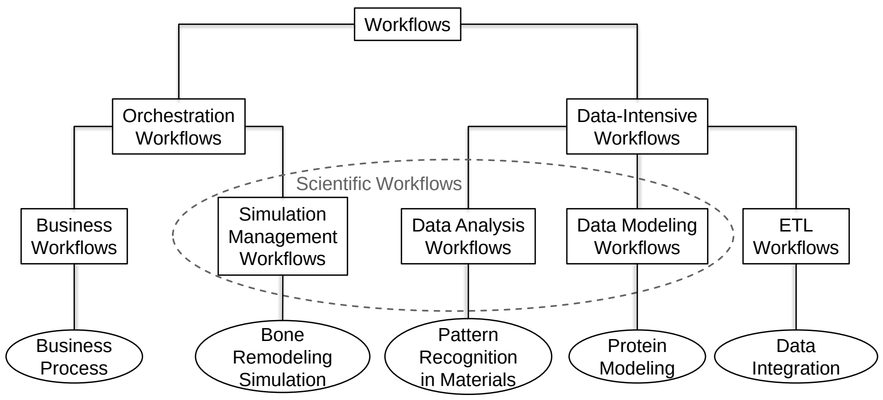

Fig. 2. Classification of workflows regarding data-intensiveness and problem they solve. Workflow classes are shown in rectangles, whereas circles state scenarios or use cases the respective classes find application in.

2. DATA PROCESSING IN DATA-INTENSIVE WORKFLOWS

In this section, we deal with major aspects and the state of the art of data processing in data-intensive workflows. We first introduce a classification of workflows regarding their data-intensiveness and the problem they solve and show the challenges of data processing in these workflows. Afterwards, we deal with different types of workflow languages that are used in such settings. We then present representative concepts for data management in workflows and relate them to the workflow classes. Based on this, we finally illustrate the main aspects by means of a sample scenario.

2.1 Workflow Classes and their Challenges for Data Processing

Figure 2 shows the classification of workflows regarding their data-intensiveness and the problem they solve. *Orchestration workflows* originate from the area of business applications where *business workflows* or production workflows orchestrate the execution of different and heterogeneous applications to realize and automate business processes [Leymann and Roller 1999]. Here, the Service Oriented Architecture (SOA), in particular the Web Services technology, is used to integrate the applications. Similarly, *simulation management workflows* coordinate the interaction with simulation applications and resources that execute calculations and data management tasks for simulation processes [Görlach et al. 2011]. See [Reimann et al. 2011] for a workflow of a bone remodeling simulation that is used to research skeletal disorders, e. g., of a human femur.

In contrast to orchestration workflows, *data-intensive workflows* treat data and their processing as first-class citizens. Such workflows typically process huge amounts of possibly distributed and heterogeneous data and carry out a multiplicity of complex data processing steps that may be directly reflected in the workflow definitions. The data may be processed on external resources or even within the workflow. A sub-class of data-intensive workflows are visualization or *data analysis workflows*. Their goal is to visualize previously generated data and/or to provide new insights from such data. Typical operations are object identification, feature discovery, and pattern recognition, e. g., in a workflow using similarity search and classification to detect the patterns among chemical compounds or other materials [Kamath et al. 2009]. *Data modeling workflows* try to find a suitable model that describes a certain problem or try to search for patterns in such models. This workflow class includes applications such as model reductions that determine a reduced and thus less compute-intensive model of computation for a simulation problem, without loosing too much precision in the computation [Haasdonk and Ohlberger 2008]. Other examples deal with modeling certain materials or searching for certain structures in material models, e. g., protein or genome modeling or analysis workflows [Berg et al. 2007], [Da Cruz et al. 2010]. Typical data management operations in such workflows

Table I. Classification of data management concepts for workflows.

Execution of Data Management \ **Definition of Data Management**	*Within Workflow*	*External to Workflow*
Within Workflow System	Integrated Case	–
External to Workflow System	Hybrid Case	Separate Case

are sampling and probing subsets of data and pattern matching. The third sub-class of data-intensive workflows constitute *extraction, transformation, and load (ETL) workflows*, i. e., ETL processes that are modeled and executed using workflow technology [Reimann et al. 2011], [Maier et al. 2005]. Their main goal is to provide a data integration or provisioning process for superordinate applications or workflows. For example, assume ETL workflows to upload data into a data warehouse or to capture and pre-process scientific data. These workflows include operations for loading or retrieving a bulk of data, filtering a data set, and joining two data sets. Recently, the term *scientific workflow* has been coined as generic term for some of these workflow classes [Taylor et al. 2007]. This includes simulation management workflows, data analysis workflows, and data modeling workflows (see Figure 2).

2.2 Workflow Languages

The different workflow classes exhibit different kinds of workflow languages. In particular, we distinguish between data-flow-oriented and control-flow-oriented languages. A data flow defines data dependencies between workflow tasks. These data dependencies correspond to unidirectional channels over which data streams are sent. Each task is assigned with input queues for buffering individual data items of such data streams. As soon as the input queues of a task are filled with a certain number or combination of data items, the task processes theses items according to its functional definition. For example, it transforms data items to another format or filters specific items. It then forwards the resulting data items to the input queues of tasks succeeding in the data flow. Such data-flow-oriented languages are frequently used to describe all kinds of data-intensive workflows. There already exist market-proven products that show specific solutions in terms of modeling languages, optimization techniques, and execution engines for data-flow-oriented problems, e. g., IBM InfoSphere Streams[1].

A control flow usually forms a directed, acyclic graph (DAG) whose nodes are the tasks and whose edges constitute causal dependencies between these tasks. Each task is executed at most once and its execution may only start after all preceding tasks within the DAG have successfully and completely finished their execution. Some workflow languages also allow their tasks to be procedural elements that each may contain another DAG of tasks as sub-workflow and that define certain execution semantics of such sub-workflows. For example, loops may define the repeated execution of the inner tasks as long as a certain condition holds [Leymann and Roller 1999]. As control-flow-oriented workflow languages are increasingly adapted to the needs of data-intensive workflows [Böhm et al. 2007], [Slominski 2007], we focus on this setting. In particular, we focus on the local data processing in such workflows. This local data processing is based on handling process variables as described in the following section.

2.3 Data Management in Workflows

Concepts for data management in workflows are characterized along two dimensions: (1) whether the data management operations are defined within the workflow or external to it and (2) whether they are executed within the workflow system or external to it (see Table I). The first concept, the *separate case*, originates from control-flow-oriented settings of SOA-based business workflows. In this spirit,

[1]IBM InfoSphere Streams: http://www-01.ibm.com/software/data/infosphere/streams/

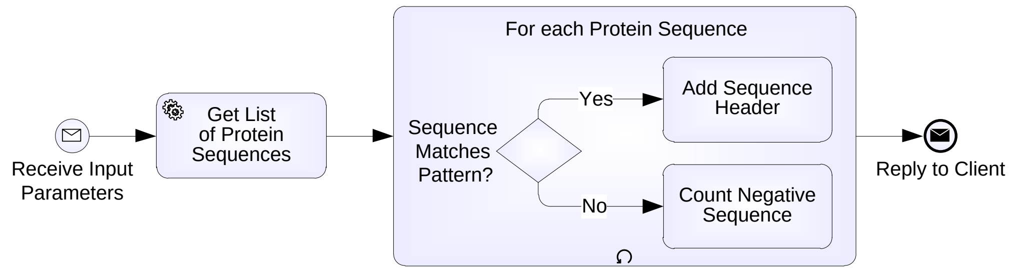

Fig. 3. Protein modeling workflow in Business Process Modeling Notation (BPMN).

services may encapsulate access to one or more data sources and provide operations on this data, e. g., for data extraction or manipulation. Such services are typically called data services or data walls as they hide the data sources and the data processing from the workflow that invokes the services. So, data processing is strictly separated from workflow processing, i. e., the data management is both defined and executed by services or external data sources. Scientific applications recently adopted the SOA paradigm as well [Taylor et al. 2007]. Most of today's scientific workflow systems offer some kind of activities to invoke services, e. g., see [Ludäscher et al. 2006], [Barga et al. 2008], and [Görlach et al. 2011], and may thus use services for data management.

The *integrated case*, on which we focus in this article, reflects local data processing within workflow systems. So, data and workflow processing are integrated together. Here, data is first shipped to the execution context of the relevant workflow, i. e., to input queues of workflow tasks in a data-flow-oriented setting or to process variables in case of control-flow-oriented languages. Workflow tasks embed or define data management operations that are executed locally on the shipped data within the workflow execution environment. As shown in Figure 1, this environment may integrate a database system or other kinds of local data processing units that store or manage the shipped data as well as its processing by the workflow.

The third concept is an approach using data management activities that directly embed data management operations. When such an activity is executed, it seamlessly accesses the specified external data source to issue its embedded operation against this source. So, this forms a *hybrid case* as data management operations are defined within workflows as in the integrated case, but are executed by external data sources as in the separate case. These data management activities may also be part of separate ETL workflows that encapsulate data integration or provisioning processes for superordinate workflows. They mainly come from business workflow solutions, e. g., the workflow products of IBM, Microsoft, and Oracle that allow for the integration of SQL statements into BPEL processes [Vrhovnik et al. 2008]. Nevertheless, the same also holds for the scientific workflow solutions Microsoft Trident [Barga et al. 2008] and Kepler [Ludäscher et al. 2006]. The latter even allows to access proprietary file systems and sensor networks. The Business Process Execution Language extension for Data Management (BPEL-DM) is a data-source-independent approach to embed any kind of data management operation for any kind of data source directly within workflows [Reimann et al. 2011].

2.4 Sample Scenario for Data-Intensive Workflows

Figure 3 shows the activities and their control flow of a workflow for protein modeling [Berg et al. 2007]. This workflow uses pattern matching to find important regions within protein sequences or families, e. g., amino acids that are relevant for a certain chemical reaction. In doing so, it identifies or investigates proteins that can solve a certain chemical or biological problem. The workflow first gets input parameters from the client such as identifiers for data sources storing the protein sequences to be investigated. Afterwards, it calls a service to retrieve a list of the relevant protein sequences. This list is stored in a process variable within the workflow. The latter then iterates over the list and searches for a certain pattern within each protein sequence, e. g., by means of a regular expression.

If a sequence matches the pattern, the workflow adds the sequence header to a list of headers in another process variable. Otherwise, it increments a counter for negative sequences. After having processed all protein sequences in this way, it sends both the number of negative sequences and the list of sequence headers of positive sequences back to the client.

According to our definition of a data-intensive workflow, the protein modeling workflow carries out several data processing steps including data retrievals, variable assignments, and pattern matching. This way, it treats data and their processing as first-class citizens. Most of the data are processed directly within the workflow, i. e., in its process variables. The data sizes may range from a few hundred KBs to several MBs or even GBs. The workflow benefits from the control-flow-oriented workflow technology in various ways. For example, we get one common level of abstraction for this workflow and for superordinate simulation management workflows that repeatedly call the protein modeling workflow in order to investigate or to model a set of protein sequences or families. This enables a comprehensive re-engineering or optimization across all involved workflows [Radeschütz and Mitschang 2009], [Vrhovnik et al. 2007], [Böhm et al. 2007]. Furthermore, we can re-use the protein modeling workflow for various simulation management workflows or other workflows in a generic way. However, using control-flow-oriented languages for this kind of workflow sometimes leads to an increased execution time compared to data-flow-oriented approaches. In the next section, we show how to improve the data processing within such workflows and thus to reduce their execution time.

3. IMPROVED DATA PROCESSING IN CONTROL-FLOW-ORIENTED WORKFLOWS

To illustrate our approach, we first sketch the main components of the typical architecture of control-flow-oriented workflow execution environments. Afterwards, we show how this architecture is extended by our approach and introduce various techniques to improve the workflow-internal data processing.

3.1 Current Architecture for Local Data Processing in Control-Flow-Oriented Workflows

Figure 4(a) shows the current architecture for control-flow-oriented workflow execution environments as they are typically used for orchestration workflows. For better readability, we leave out components that are not directly relevant for the workflow-internal data processing, e. g., a compiler or deployment component for workflow models. Such environments contain a local database system (DBS). Remark that this is not the type of external database stores a workflow may access via the separate and hybrid cases of data management (see Table I). In fact, this local DBS supports the integrated case of data management and serves as persistent data store for the workflow-local data in process variables and for metadata such as auditing information. However, it does not process this data, e. g., during variable assignments. Instead, the *workflow runtime* component itself performs this data processing. For that purpose, it contains a *pool of variables* that manages all process variables and their contents, e. g., realized as a heap of Java objects, and an *expression evaluation engine* that evaluates expressions, e. g., XPath expressions, on these variables. A *data processing logic* component defines how all data-processing workflow activities and constructs work on the variables and how the expression evaluation engine as well as the pool of variables are employed. For example, this data processing logic controls the data processing for variable assignments, service calls, and control flow decisions.

A *persistence manager*, e. g., a Data Access Object (DAO) layer, negotiates between the runtime and the local DBS and manages the persistence of the workflow-local data, i. e., it stores and loads the data in or from the DBS either automatically or when the runtime triggers it. To manage the variable contents, the database contains a *pool of variables* as well. The persistence manager provides the mapping between the variables in the pool of the workflow runtime and those in the pool of the database. This way, the runtime is independent from the concrete local DBS, i. e., we can exchange this DBS quite easily. The database management system contains a *query/expression execution engine*, but this engine is not used to execute the workflow-internal data processing. It just gets simple queries from the persistence manager to load or store whole variable contents.

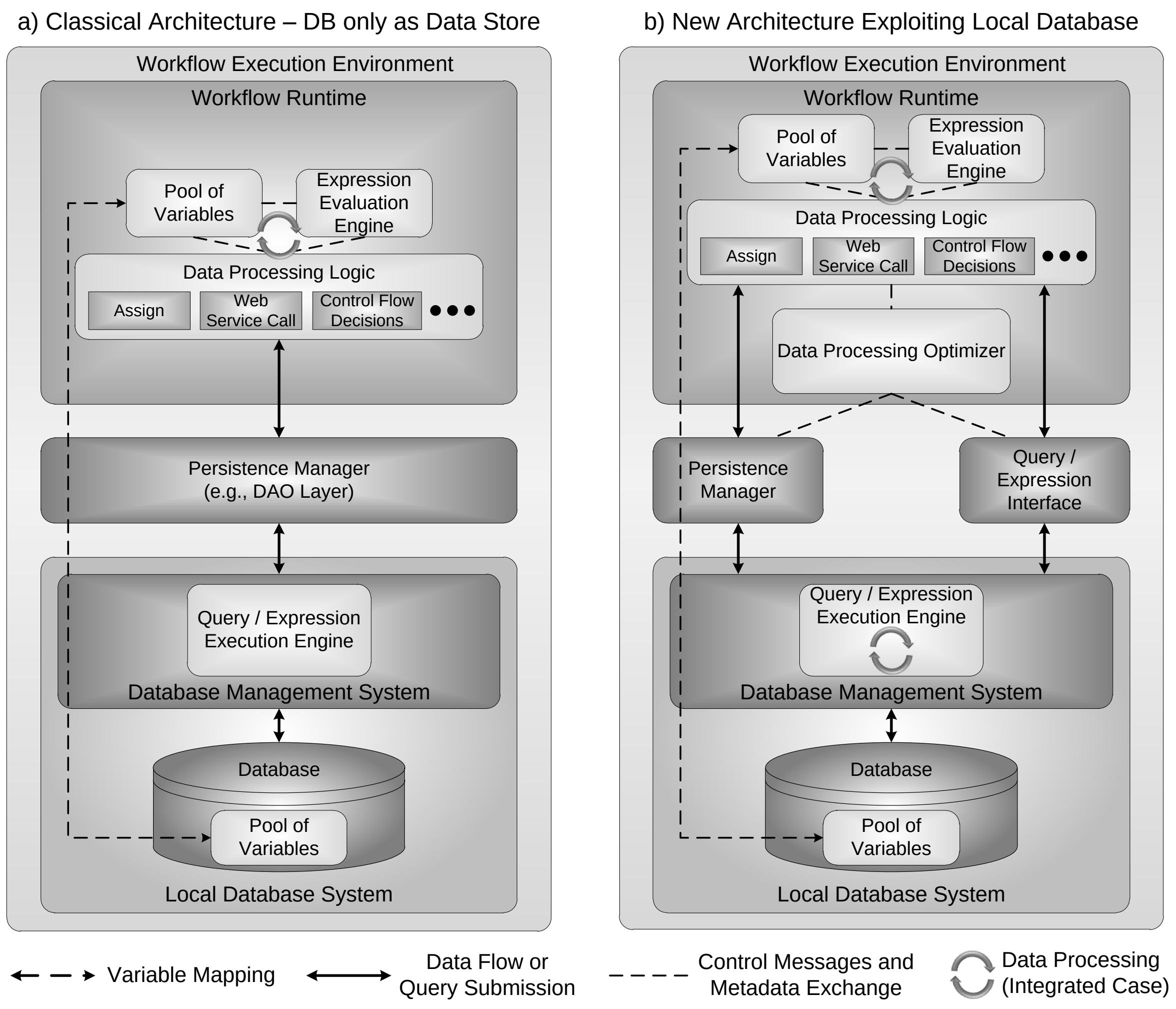
a) Classical Architecture – DB only as Data Store
Workflow Execution Environment
Workflow Runtime
Pool of Variables
Expression Evaluation Engine
Data Processing Logic
Assign
Web Service Call
Control Flow Decisions
Persistence Manager (e.g., DAO Layer)
Query / Expression Execution Engine
Database Management System
Database
Pool of Variables
Local Database System
b) New Architecture Exploiting Local Database
Workflow Execution Environment
Workflow Runtime
Pool of Variables
Expression Evaluation Engine
Data Processing Logic
Assign
Web Service Call
Control Flow Decisions
Data Processing Optimizer
Persistence Manager
Query / Expression Interface
Query / Expression Execution Engine
Database Management System
Database
Pool of Variables
Local Database System
Variable Mapping
Data Flow or Query Submission
Control Messages and Metadata Exchange
Data Processing (Integrated Case)

Fig. 4. Current architecture for data processing in workflows (a) and architecture for an improved data processing (b).

3.2 A New Architecture for Local Data Processing

To ensure a reliable and efficient local data processing in data-intensive workflows described in control-flow-oriented languages, we have extended this architecture by components that provide an improved workflow-internal data processing (Figure 4(b)). The data processing tasks to be performed during workflow execution are partitioned between the workflow runtime and the local DBS in order to exploit their respective capabilities as much as possible. This is transparent to the workflow modeler, i. e., it does not change workflow models, but only influences the execution of such models. Beside the persistence manager, the *query/expression interface* allows to push down workflow-internal data processing operations from the runtime to the local DBS that finally executes these operations. Using this component, the runtime only issues queries or expressions against the DBS and receives small data items, e. g., Boolean values needed for control flow decisions. As the queries to execute the data processing operations may vary between different DBSs, the query/expression interface has to store query templates for concrete DBSs and has to set the necessary parameter values within these query templates. This way, the workflow runtime is again independent from the concrete local DBS.

The *data processing optimizer*, embedded in the workflow runtime, controls the data processing logic and decides whether the workflow-internal data processing operations are assigned to the

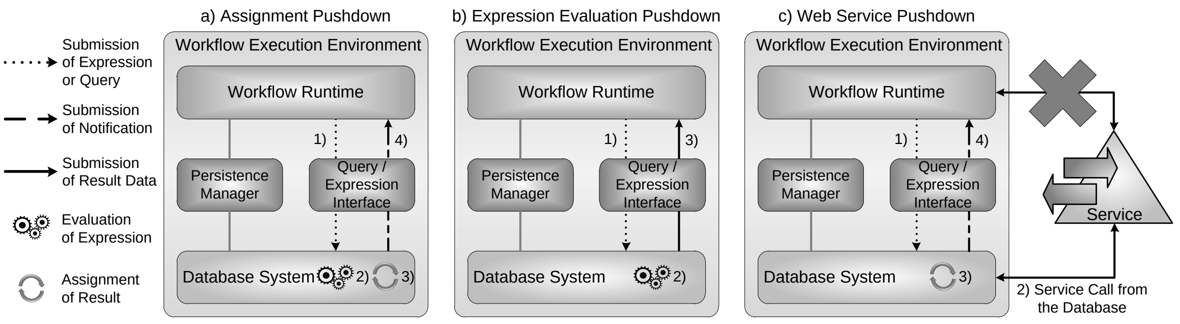
a) Assignment Pushdown
b) Expression Evaluation Pushdown
c) Web Service Pushdown
Submission of Expression or Query
Submission of Notification
Submission of Result Data
Evaluation of Expression
Assignment of Result
Workflow Execution Environment
Workflow Runtime
Persistence Manager
Query / Expression Interface
Database System
1)
2)
3)
4)
Service
2) Service Call from the Database

Fig. 5. Assignment Pushdown (a), Expression Evaluation Pushdown (b), and Web Service Pushdown (c). Numbers indicate the orders in which the individual steps are carried out.

workflow runtime or to the local DBS and in which of both components the corresponding data is stored. The main goals of this optimization are to decrease the amount of data transfered between the runtime and the DBS as well as to exploit the mature data processing facilities of the DBS and its query/expression execution engine whenever appropriate. The decisions mainly depend on the capabilities of the runtime and the local DBS, the data sizes, and the complexities of the involved queries and expressions. All these optimization criteria are described by metadata the optimizer retrieves from the respective components. The optimizer can make its decisions in several granularities, e. g., the same decision for all instances of one workflow model, for individual instances or sets of instances, for single data processing tasks, or for single process variables or sets of variables.

3.3 Techniques to Improve Workflow-Internal Data Processing

The data processing optimizer may employ various techniques that partition the workflow-internal data processing operations between the workflow runtime and the local DBS. Some of them are depicted in Figure 5. The *Assignment Pushdown* shifts the responsibility for executing process variable assignments from the workflow runtime to the DBS. The DBS receives (step 1) and evaluates (step 2) assignment expressions, e. g., XPath expressions, and assigns the expression results to the target variables (step 3). Afterwards, the runtime gets a notification whether the assignment has been executed successfully or not (step 4). The *Expression Evaluation Pushdown* is used for control flow decisions, such as transition conditions or loops. It again evaluates expressions within the DBS, but synchronously forwards the expression results back to the runtime for further processing.

Using these two techniques, the data transferred from the local DBS to the runtime are only small notifications or data items, possibly leading to performance improvements due to reduced costs for data transmission. However, the input data of the corresponding expressions, which may become big in case of data-intensive workflows, should be available in the DBS before the respective variable assignments or control flow decisions are carried out. Otherwise, it would be necessary to transfer these data from the runtime to the DBS via the persistence manager, which might lead to performance degradations. To tackle this problem, we have developed two further techniques.

A control-flow-oriented workflow may get two kinds of data that eventually have to be stored in the local DBS: (1) data as result of a service call or sent by the client and (2) literal values, e. g., XML snippets, which are defined in the workflow model and assigned to a process variable during workflow execution. The *Web Service Pushdown* deals with the first kind of data. It calls Web Services directly from the DBS instead from the runtime, thereby storing the result data of the service in the database without the indirection over the runtime and the persistence manager (see Figure 5(c)). The same principle can be applied for asynchronous communication, i. e., when a workflow sends data to a service or client or when it gets data from them, without expecting any result on either side. The literal values are already known at modeling and deployment time as they are a direct part of the

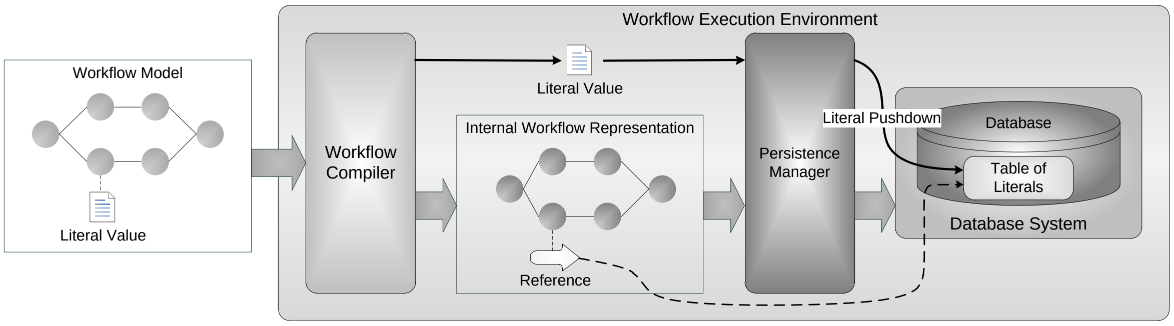

Fig. 6. Literal Pushdown.

workflow model. The *Literal Pushdown* exploits this characteristic and stores such literal values in a designated table in the DBS during the deployment phase of the relevant workflow model (Figure 6). Furthermore, it writes a reference into the internal representation of this model or into the contents of the respective variables to refer to the previously stored literal values during workflow execution.

4. BENEFITS OF THE APPROACH

In this section, we discuss the benefits of our approach. This covers its optimization potential for workflow-internal data processing, generality issues, and additional opportunities to extend current workflow execution environments by further data management functionalities.

4.1 Optimization Potential for Workflow-Internal Data Processing

In previous control-flow-oriented workflow environments, the workflow-internal data processing could only be executed by the workflow runtime. The data processing optimizer provides a new option to shift data processing to the integrated DBS. This offers a great potential for improved performance and reliability of the workflow-internal data processing in data-intensive workflows. The main reason is that the mature database technology can more efficiently and reliably deal with large volumes of data and the complex as well as frequent data processing operations that are involved in such workflows. Furthermore, we can reduce the amount of data transfered from the runtime to the local DBS because all data may be directly stored in the DBS without the indirection over the runtime.

When multiple workflow instances run concurrently, the execution of their local data processing within the workflow runtime typically leads to an increased main memory consumption of this runtime. It may get too busy or even overloaded. In such a case, the runtime pauses some of the workflow instances and stores all relevant data in the DBS in order to resume the paused instances at a later point in time. This leads to an increased execution time for some workflow instances and to reduced overall throughput. The database technology is a mature technology regarding concurrent execution of processes and regarding an efficient main memory allocation. Following our approach, we can push parts of the workflow processing to the local DBS. This may lead to a more balanced utilization of all involved system components, in particular to a more efficient overall main memory utilization, and thus to a faster execution and a higher throughput of concurrent workflow instances. Furthermore, the more efficient main memory utilization may reduce the number of main memory overloads in the workflow system and thus enhance the reliability of workflow execution.

On the other hand, processing data in the main memory of the workflow runtime leads to less costs of data access and the runtime may exploit customized internal data structures that are tailor-made to the workflow-internal data processing. For less data-intensive workflows as well as for small data sizes, e. g., for orchestration workflows, the data processing optimizer may decide to leave the responsibility for data processing to the workflow runtime. To avoid the additional overhead for this optimization decision at workflow execution time, this decision may be made during deployment time. In Section 5, we evaluate some of the arguments discussed in this section.

4.2 Generic Workflow Management

Due to the above-mentioned optimization potential, our approach broadens the set of data-intensive workflows that may be described in terms of control flow. Since this approach is also applicable to small or less data-intensive workflows as well as for small data sizes and since we extend a common architecture for control-flow-oriented workflow environments, it is suitable for orchestration workflows as well. This results in a common level of abstraction, i. e., the control flow, for all kinds of workflows, which entails several benefits. The common level of abstraction enables further optimizations over the whole spectrum from the workflow level to the data level and across all workflow classes [Vrhovnik et al. 2007], [Böhm et al. 2007]. Furthermore, it leads to an integrated tool support, i. e., all kinds of processes can be modeled using standard languages such as BPEL and executed within common tools or environments. Altogether, this forms a generic solution being applicable to all workflow classes depicted in Figure 2, in multiple domains of scientific or business applications, and even in the combination of scientific and business applications [Reimann et al. 2011], [Janowski et al. 2011].

4.3 Increased Extensibility of Workflow Execution Environments

The new possibility to intensively use the local database system in our new architecture of workflow execution environments enables a highly increased extensibility of such environments by additional data management functionalities. For example, current database systems offer useful capabilities for workflows that a workflow runtime does not offer or for which it would have to be extended in a complicated way. A geographic information system may support operations tailor-made to spatial problems. Such operations can be used in a simulation that determines spatial changes of the structure of a car in a crash test [Janowski et al. 2011]. Furthermore, we can easily extend the new architecture to support global data structures that may serve as a cache for data needed by several workflow instances or for data that is used for synchronization purposes. Traditional approaches define such global data structures in the local DBS that workflows access following the separate or hybrid approaches depicted in Table I. In contrast, the extended architecture allows to provide some kind of shared variables. The main advantage is that these shared variables are modeled and used transparently, i. e., in the same way as conventional process variables. The local DBS already offers functionality needed for the management of shared variables. This includes guaranteed persistence and life-cycle management of variables and their contents as well as concurrency control mechanisms that are needed when multiple workflow instances access the same shared variables.

Many data-intensive workflows integrate highly heterogeneous external data and load parts of them into their workflow context. Our approach serves as starting point to employ different kinds of data processing facilities for the workflow-internal data processing that are tailor-made to specific workflows. For example, some workflows might benefit from facilities for semi-structured or unstructured data such as key-value stores. For data or workflows not having high persistence requirements, we may also employ main-memory-based database systems to further improve performance. Furthermore, we can use data integration solutions to describe external and local data in a transparent way.

5. EVALUATION OF TECHNIQUES FOR WORKFLOW-INTERNAL DATA PROCESSING

Now, we evaluate the effectiveness of the techniques introduced in Section 3.3 via experiments, in particular via the protein modeling workflow of Section 2.4. In doing so, we evaluate some of the arguments discussed in the previous section regarding improved efficiency and reliability for data-intensive workflow execution.

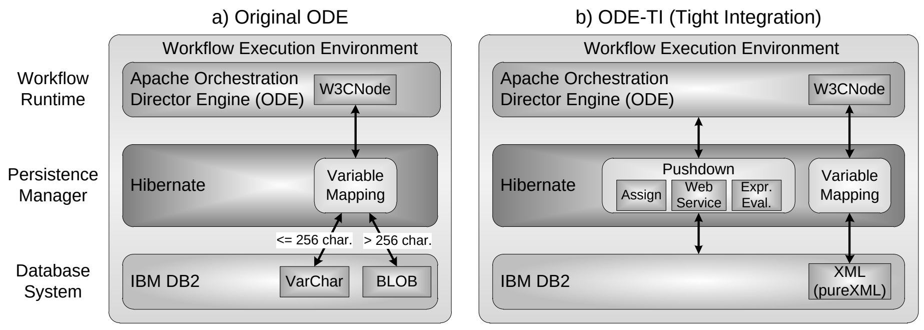

Fig. 7. Prototypes original ODE (a) and ODE-TI (b), corresponding to architectures shown in Figure 4(a) and 4(b).

5.1 Prototype and Experimental Setup

For our evaluation, we have developed two BPEL-based prototypes of a workflow execution environment. The first one reflects the current architecture of such environments as shown in Figure 4(a). We use the open source workflow engine Apache Orchestration Director Engine[2] (ODE) V1.3.4 as workflow runtime, the IBM DB2[3] V9.7 as DBS, and Hibernate[4] V3.2.5 as persistence manager (see Figure 7(a)). ODE represents variable contents as XML documents. These documents are managed as Java objects of the type W3C Node[5]. When storing the data in the DBS, Hibernate maps small XML documents to variable character fields (varchar) and larger ones to binary large objects (BLOBs). These data structures are appropriate for the original usage pattern of the local DBS, i. e., it only serves as data store for variable contents. In the following, we call this prototype *original ODE*.

To implement the extended architecture presented in Figure 4(b), we have changed the DBS-internal data structure and the corresponding mapping of Hibernate to a native XML data type in order to employ the XML-enabled query processing functionality of the DB2 pureXML technology[6] (see Figure 7(b)). We abstained from the data processing optimizer. Instead, we have extended Hibernate to enforce the techniques Assignment Pushdown, Expression Evaluation Pushdown, and Web Service Pushdown introduced in Section 3.3. This leads to a tight integration of the database and the workflow engine. We call the resulting prototype *ODE-TI* (Tight Integration).

All system components ran on a 32-bit Windows Server 2003 Enterprise Edition (Service Pack 2) operating system with two Intel Xeon PowerEdge 2850 3.2 GHz processors and 8 GB main memory. With this system environment and the above-mentioned prototypes, we have investigated whether there exist break-even points when ODE-TI is more efficient and reliable than original ODE. Criteria for such break-even points are the data size, complexity of involved expressions, and complexity of workflows. In real scenarios, concrete break-even points for decisions of the data processing optimizer are system- and tool-dependent. We have analyzed the effectiveness of the techniques introduced in Section 3.3 in two scenarios. The first one comprises small BPEL processes that facilitated the test of individual workflow activities, i. e., a BPEL *assign* activity for the Assignment Pushdown, an *if* activity for the Expression Evaluation Pushdown, and an *invoke* activity for the Web Service Pushdown. This allows us to evaluate the techniques in isolation. The second test scenario is the sample scenario of a data-intensive protein modeling workflow described in Section 2.4 that contains all these activities and runs most of them in a loop. Activity *Get List of Protein Sequences* is defined as *invoke* activity, the pattern matching step as *if* activity, and activities *Add Sequence Header* and *Count Negative* as *assign* activities. This allows us to evaluate all techniques in combination.

[2]Apache ODE: http://ode.apache.org/
[3]IBM DB2: http://www-01.ibm.com/software/data/db2/
[4]Hibernate: http://docs.jboss.org/hibernate/core/3.5/reference/en/html/
[5]W3C Node: http://download.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/Node.html
[6]IBM DB2 pureXML: http://www-01.ibm.com/software/data/db2/xml/

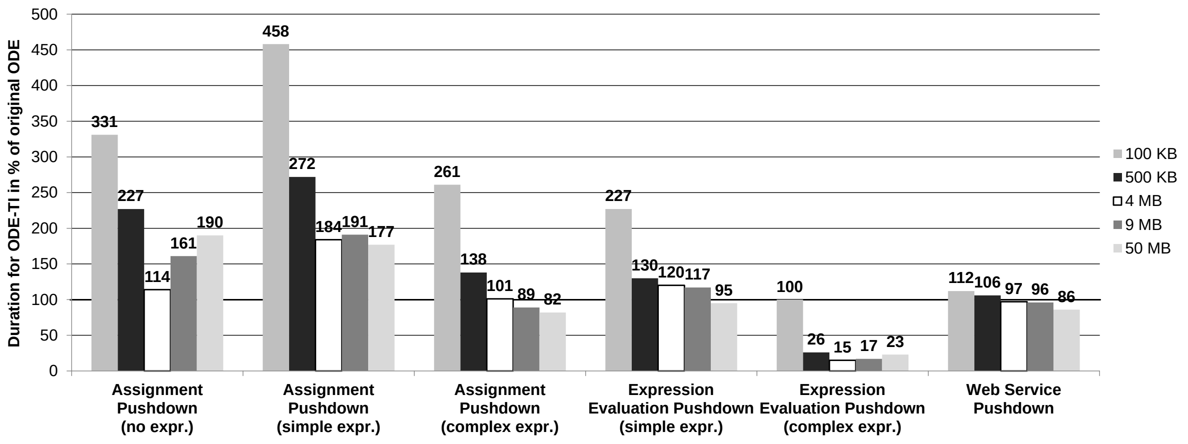

Fig. 8. Effectiveness of techniques for workflow-internal data processing.

The underlying data set of both test scenarios is the list of protein sequences of the protein modeling workflow. Each protein sequence is organized as XML element contained in an XML sequence of such elements. The tests have been carried out for five different data sizes: 100 KB, 500 KB, 4 MB, 9 MB, and 50 MB. This corresponds to 40, 199, 697, 1394, and 7695 protein sequences, respectively, as well as the same number of iterations of the inner loop. Larger XML documents could not be tested since Apache ODE either causes main memory overloads for such data sets or generally stops workflow execution after 2000 seconds without result or failure notification. Remark that this is no severe limitation of our experimental setting as for performance reasons, processing such large amounts of data should be assigned to external resources anyway (see Figure 1).

5.2 Test Results

Up to 9 MB, we have carried out each BPEL process of the first test scenario 100 times, and 50 times for 50 MB. Afterwards, we have calculated the mean average of the relevant activity durations. The durations of original ODE are taken as 100%, wheras those of ODE-TI are reported in Figure 8 in relation to those of original ODE. For the Assignment Pushdown, we have used three different complexities of the involved XPath expressions. The underlying *assign* activity first carries out a read operation to evaluate this expression on the input list of protein sequences, followed by a write operation that stores the expression result to a variable. The first expression complexity involves *no expression*, i. e., it stores the whole input data. The second one contains an expression to select exactly one protein sequence (*simple expression*), and the third one concatenates two sequences (*complex expression*). So, the write operation in these two cases stores a data set with constant size, i. e., its size does not depend on the size of the input data. Regarding no or a simple expression, the Assignment Pushdown shows an average performance degradation for every data size that ranges between 358% and 11%. For the complex expression, it shows degradations by 161% and 38% for 100 KB and 500 KB. When increasing the data size to more than 4 MB, we reach a break-even point in the used system environment where ODE-TI gets slightly better than original ODE by up to 18%.

In the same way, we tested the Expression Evaluation Pushdown with a simple expression that accesses one protein sequence and a complex expression that accesses two of them. The underlying *if* activity only performs a read operation, and no subsequent write operation. For the simple expression, ODE-TI shows performance degradations from 17% to 127% when reading data up to 9 MB. However, the break-even point is reached for 50 MB, where we get an improvement of 5%. When testing the complex expression, we get the same performance for 100 KB and improvements between 74% and 85% for the other data sizes. The results of the Web Service Pushdown get better with increasing data size and range from slight degradations of 12% to slight improvements up to 14%.

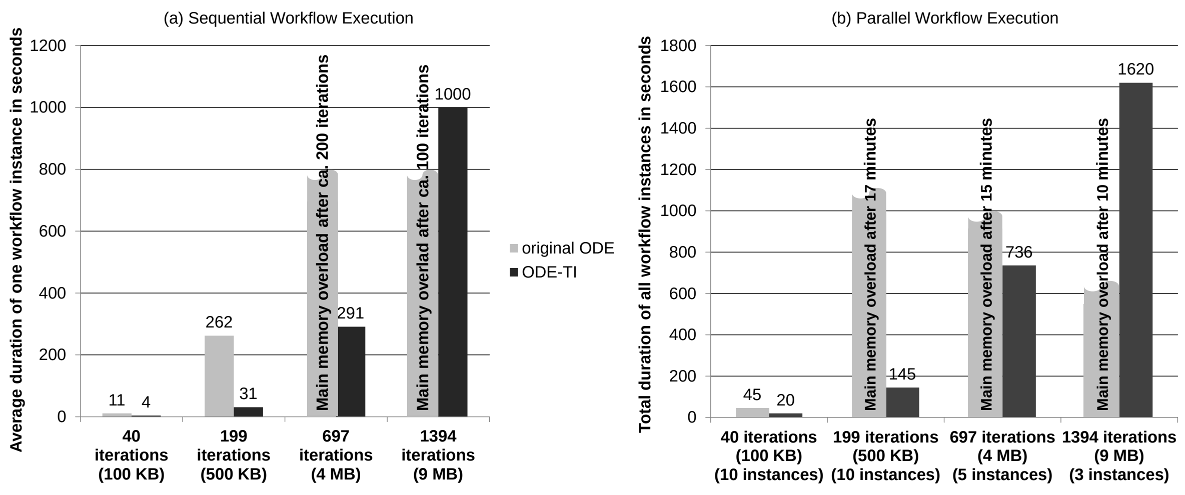

Fig. 9. Performance improvements for the protein modeling workflow.

In summary and as described in Section 4.1, these experiments show that our techniques considered in isolation result in performance improvements mainly for complex expressions that are evaluated on large data sets.

Comparing the results of the Expression Evaluation Pushdown and the Assignment Pushdown and considering that the only relevant difference between the underlying *if* and *assign* activities is the additional write operation of the *assign*, we see that the workflow runtime in original ODE deals more efficiently with write operations than the database system in ODE-TI. The main reason for the additional overhead in the database system is that it needs to store log information on disk for the involved data manipulation queries. Furthermore, it needs to adapt indexes when new data is inserted or when existing data is updated. In case of the Assignment Pushdown without expression, the results first get better with an increasing data size. However, this turns into the opposite when crossing the 4 MB size. This is because for bigger data sets the effects of the write operation in the database system superimpose those of the read operation. All other test cases involving an expression either do not perform a write operation or the write operation stores data of constant size. Hence, the effects of write operations get less important with growing data sets and ODE-TI achieves better results the bigger the input data sets are. In original ODE, the *invoke* activity regarding the Web Service Pushdown stores the result of the service in the local DBS for persistence and recovery purposes. This constitutes additional costs for transferring data from the workflow runtime to the local DBS, which are circumvented in ODE-TI by the Web Service Pushdown (see Section 4.1). These transfer costs in original ODE largely compensate the above-mentioned overhead for writing the service result in the database system of ODE-TI, but they cross this overhead when the data size increases.

For the second test scenario, we have executed the protein modeling workflow either in sequence or in parallel. For the sequential execution, we have carried out 100 instances for both 100 KB and 500 KB. The 4 MB and 9 MB cases have been executed 50 times. The 50 MB and larger documents could not be tested due to a main memory overload in original ODE and workflow execution times above 2 million ms in ODE-TI as described in Section 5.1. Figure 9(a) shows the absolute values of the average duration of one workflow instance compared between original ODE and ODE-TI. These results indicate the potential of our approach: For 100 KB, we can reduce the duration by nearly a factor of 3, and even by a factor of more than 8 using 500 KB. For 4 and 9 MB, original ODE was not able to execute the entire workflow at least once. Due to a main memory overload in the workflow runtime, it crashed after approximately 200 or 100 iterations, respectively. However, ODE-TI successfully executed all 50 instances of the workflow for both data sizes. Altogether, these results correspond to the argument of Section 4.1 that the database technology can more efficiently and reliably deal with high volumes of data and complex as well as frequent data processing operations involved in data-intensive workflows. The standard version of Apache ODE comes with a Java-based and embedded Apache Derby[7] database system. When using Derby instead of DB2 in original ODE, the problem with main memory overloads would even be worsened since Derby would be executed in the same Java process as ODE and would thus share its main memory.

Figure 9(b) shows the results for parallel workflow execution where all instances are started at the same time. Here, we have executed the 100 KB and 500 KB cases ten times, the 4 MB case five times, and three instances for 9 MB. For each data size, the figure reports the total durations of all executed workflow instances. ODE-TI outperforms original ODE in any of these cases. It reduces the total duration by a factor of more than 2 already for 100 KB. When the data size grows, original ODE gets problems to execute at least one of the workflow instances. After ten to 17 minutes, it aborts with a main memory overload again. ODE-TI was able to execute all instances in 145 seconds for 500 KB, 736 seconds for 4 MB, and 1620 seconds in the 9 MB case. In summary and as described in Section 4.1, the evaluated techniques lead to improved efficiency and reliability for both sequential and parallel execution of data-intensive workflows such as the protein modeling workflow.

6. RELATED WORK

Previous work related to the approach we described in this article mainly deals with two aspects: (1) various optimization opportunities for the data management in workflows and (2) the modeling of data-intensive workflows. In the following, we highlight major work in these areas.

6.1 Optimization of Data Management in Workflows

For both data-flow-oriented and control-flow-oriented workflow languages, there exist several optimization opportunities for the data management in workflows. Most of them are geared to a certain case of data management as depicted in Table I. For example, workflow systems that allow for binding services at workflow runtime focus on the optimization of data management in the separate case, but not the integrated one as in our approach [Karastoyanova et al. 2007]. Further optimization approaches represent rule-based techniques for re-engineering workflow models, which are mainly applied during the phases workflow design or workflow deployment. In [Radeschütz and Mitschang 2009], for example, the authors present an extended data warehouse approach that integrates workflow-related data and operational business data to support extended analysis techniques for a comprehensive business process optimization. The approach presented in [Vrhovnik et al. 2007] is mainly suited for the hybrid case of Table I as it focuses on the optimization of workflows with embedded SQL statements that are sent to external database systems. The techniques introduced in [Böhm et al. 2007], which are applied during workflow execution phase, can be used for optimizing data integration workflows that reflect the integrated case of Table I. All these rule-based re-engineering approaches have in common that they change the structure of workflow definitions. In contrast, our optimization approach for the integrated case increases the value of these approaches as it adds a new kind of optimization. This optimization is transparent to the workflow definitions, i. e., it does not change them, but only influences their execution within the system architecture of the workflow execution environment.

Data-flow-oriented workflow languages may employ additional optimization techniques for the integrated case of data management. In particular, this encompasses pipeline parallelism between workflow tasks [Ludäscher et al. 2009] and the possibility to employ scalable and parallel data processing infrastructures such as MapReduce or high performance computing (HPC) environments [Zinn et al. 2010], [Coutinho et al. 2010]. However, these techniques cannot be easily adopted in control-flow-oriented languages as they strictly separate between the process logic and the data or application processing.

[7]Apache Derby: http://db.apache.org/derby/

6.2 Modeling of Data-Intensive Workflows

The individual workflow classes depicted in Figure 2 typically either employ a rather data-flow-oriented or control-flow-oriented workflow language. Orchestration workflows are largely described in terms of control flow [Leymann and Roller 1999], [Görlach et al. 2011]. In contrast, most current data-intensive workflows are modeled using pure data-flow-oriented languages. The main reasons are that they treat data and their processing as first-class citizens and that this offers the additional optimization opportunities described above. Nevertheless, control-flow-oriented workflow languages are increasingly adapted to the needs of data-intensive workflows [Böhm et al. 2007], [Slominski 2007]. The main reasons mentioned in Section 4.2 are the resulting common level of abstraction for all kinds of workflows, the corresponding possibility for a comprehensive workflow optimization, and generality issues. Further benefits include the sophisticated fault, compensation, and event handling capabilities at the workflow level, as well as the support for transaction concepts, a persistent execution state, and recovery of workflows [Akram et al. 2006]. In addition, many control-flow-based workflow systems offer further useful capabilities, e. g., user interactions or workflow monitoring. In summary, the adoption of control-flow-oriented workflow languages for data-intensive workflows may significantly improve the modeling and execution of such workflows.

7. CONCLUSION AND FUTURE WORK

In this article, we introduced a generic and extensible approach to improve the local data processing in control-flow-oriented workflow execution environments. This approach is targeted at data-intensive workflows, but it is also applicable to other kinds of workflows in multiple scientific or business domains. It includes various techniques to partition the local data processing tasks to be performed during workflow execution in an improved way. These tasks are either assigned to the workflow runtime or to the tightly integrated local database engine. The techniques encompass the execution of variable assignments, expression evaluations for control flow decisions, and Web Service calls by the database engine, as well as storing literal values in this database during workflow deployment. Based on our ODE-TI prototype, which enforces these techniques, we evaluated their effectiveness by means of small test workflows and a more complex and data-intensive protein modeling workflow. The test results demonstrated that the techniques improve the efficiency and reliability of the local data processing in data-intensive workflows. So, our approach broadens the set of such workflows that may be described in terms of control flow.

Future work will try to extend the scalability of our ODE-TI prototype with respect to larger XML document sizes. Furthermore, we will employ main-memory-based techniques to further improve performance. This might even entail that ODE-TI may already be used for smaller data sizes and for less complex expressions and workflows.

REFERENCES

Akram, A., Meredith, D., and Allan, R. Evaluation of BPEL to Scientific Workflows. In *Proc. of the 6th International Symposium on Cluster Computing and the Grid*. Washington, DC, USA, pp. 269–274, 2006.

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., and Simmhan, Y. The Trident Scientific Workflow Workbench. In *Proc. of the 4th International Conference on e-Science*. Indianapolis, IN, USA, pp. 317–318, 2008.

Berg, J. M., Tymoczko, J. L., and Stryer, L. *Biochemistry*. W. H. Freeman and Co., New York City, NY, USA, 2007.

Böhm, M., Habich, D., Wloka, U., Bittner, J., and Lehner, W. Towards Self-Optimization of Message Transformation Processes. In *Communications of the 11th East-European Conference on Advances in Databases and Information Systems (ADBIS 2007)*. Varna, Bulgaria, pp. 116–125, 2007.

Coutinho, F., Ogasawara, E., de Oliveira, D., Braganholo, V., Lima, A., Dávila, A., and Mattoso, M. Data Parallelism in Bioinformatics Workflows using Hydra. In *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*. Chicago, IL, USA, pp. 507–515, 2010.

Da Cruz, S., Batista, V., Silva, E., Tosta, F., Vilela, C., Cuadrat, R., Tschoeke, D., Davila, A., Campos, M., and Mattoso, M. Detecting Distant Homologies on Protozoans Metabolic Pathways using Scientific Workflows. *International Journal of Data Mining and Bioinformatics (IJDMB)* 4 (3): 256–280, 2010.

Deelman, E. and Chervenak, A. Data Management Challenges of Data-Intensive Scientific Workflows. In *Proc. of the 8th International Symposium on Cluster Computing and the Grid*. Washington, DC, USA, pp. 687–692, 2008.

Freire, J., Koop, D., Santos, E., and Silva, C. T. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering* 10 (3): 11–21, 2008.

Görlach, K., Karastoyanova, D., Leymann, F., Reiter, M., and Sonntag, M. Conventional Workflow Technology for Scientific Simulation. In *Guide to e-Science*, Y. Yang, L. Wang, and W. Jie (Eds.). Springer, London, UK, chapter 11, pp. 323–352, 2011.

Haasdonk, B. and Ohlberger, M. Reduced Basis Method for Finite Volume Approximations of Parametrized Linear Evolution Equations. *Mathematical Modelling and Numerical Analysis* 42 (2): 277–302, 2008.

Janowski, P., Mitschang, B., and Gollmann, A. Issues and Characteristics of Testing as Part of the Design Process in Mechanical Engineering. In *Proc. of the 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2011)*. Lausanne, Switzerland, pp. 599–604, 2011.

Kamath, C., Wale, N., Karypis, G., Pandey, G., Kumar, V., Rajan, K., Samatova, N. F., Breimyer, P., Kora, G., Pan, C., and Yoginath, S. Scientific Data Analysis. In *Scientific Data Management: Challenges, Technology, and Deployment*, A. Shoshani and D. Rotem (Eds.). Computational Science Series. Chapman & Hall, chapter 8, pp. 281–323, 2009.

Karastoyanova, D., Wetzstein, B., van Lessen, T., Wutke, D., Nitzsche, J., and Leymann, F. Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware. In *Proc. of the 2nd International ICDE Workshop on Service Engineering (SEIW 2007)*. Istanbul, Turkey, pp. 347–354, 2007.

Leymann, F. and Roller, D. *Production Workflow: Concepts and Techniques*. Prentice Hall, Englewood Cliffs, 1999.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience* 18 (10): 1039–1065, 2006.

Ludäscher, B., Altintas, I., Bowers, S., Cummings, J., Critchlow, T., Deelman, E., Roure, D. D., Freire, J., Goble, C., Jones, M., Klasky, S., McPhillips, T., Podhorszki, N., Silva, C., Taylor, I., and Vouk, M. Scientific Process Automation and Workflow Management. In *Scientific Data Management: Challenges, Technology, and Deployment*, A. Shoshani and D. Rotem (Eds.). Computational Science Series. Chapman & Hall, chapter 13, pp. 467–508, 2009.

Ludäscher, B., Weske, M., Mcphillips, T., and Bowers, S. Scientific Workflows: Business as Usual? In *Proc. of the 7th International Conference on Business Process Management (BPM 2009)*. Ulm, Germany, pp. 31–47, 2009.

Maier, A., Mitschang, B., and Leymann, F. On Combining Business Process Integration and ETL Technologies. In *Gesellschaft für Informatik (ed.): Datenbanksysteme für Business, Technologie und Web*. Karlsruhe, Germany, pp. 533–546, 2005.

Radeschütz, S. and Mitschang, B. Extended Analysis Techniques for a Comprehensive Business Process Optimization. In *Proc. of the International Conference on Knowledge Management and Information Sharing (KMIS 2009)*. Madeira, Portugal, pp. 77–82, 2009.

Reimann, P., Reiter, M., Schwarz, H., Karastoyanova, D., and Leymann, F. SIMPL - A Framework for Accessing External Data in Simulation Workflows. In *Gesellschaft für Informatik (ed.): Datenbanksysteme für Business, Technologie und Web*. Kaiserslautern, Germany, pp. 534–553, 2011.

Slominski, A. Adapting BPEL to Scientific Workflows. In *Workflows for e-Science - Scientific Workflows for Grids*, I. Taylor, E. Deelman, and D. Gannon (Eds.). Springer, London, UK, chapter 14, pp. 208–226, 2007.

Taylor, I., Deelman, E., and Gannon, D. *Workflows for e-Science - Scientific Workflows for Grids*. Springer, London, UK, 2007.

Vrhovnik, M., Schwarz, H., Radeschütz, S., and Mitschang, B. An Overview of SQL Support in Workflow Products. In *Proc. of the 24th International Conference on Data Engineering (ICDE 2008)*. Cancùn, México, pp. 1287–1296, 2008.

Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., and Kraft, T. An Approach to Optimize Data Processing in Business Processes. In *Proc. of the 33rd International Conference on Very Large Data Bases (VLDB 2007)*. Vienna, Austria, pp. 615–626, 2007.

Zinn, D., Bowers, S., Köhler, S., and Ludäscher, B. Parallelizing XML Data-Streaming Workflows via MapReduce. *Journal of Computer and System Sciences* 76 (6): 447–463, 2010.