Dirk Habich, Sebastian Richly, Uwe Assmann, Wolfgang Lehner

# Using Cloud Technologies to Optimize Data-Intensive Service Applications

**Erstveröffentlichung in / First published in:**

*IEEE 3rd International Conference on Cloud Computing*. Miami, 05.–10.07.2010. IEEE, S. 19-26. ISBN 978-1-4244-8207-8

DOI: https://doi.org/10.1109/CLOUD.2010.56

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

# Using Cloud Technologies to Optimize Data-Intensive Service Applications

Dirk Habich, Wolfgang Lehner
*Dresden University of Technology*
*Database Technology Group*
*Dresden, Germany*
{*dirk.habich,wolfgang.lehner*}*@tu-dresden.de*

Sebastian Richly, Uwe Assmann
*Dresden University of Technology*
*Software Engineering Group*
*Dresden, Germany*
{*sebastian.richly,uwe.assmann*}*@tu-dresden.de*

***Abstract*—The role of data analytics increases in several application domains to cope with the large amount of captured data. Generally, data analytics are data-intensive processes, whose efficient execution is a challenging task. Each process consists of a collection of related structured activities, where huge data sets have to be exchanged between several loosely coupled services. The implementation of such processes in a service-oriented environment offers some advantages, but the efficient realization of data flows is difficult. Therefore, we use this paper to propose a novel SOA-aware approach with a special focus on the data flow. The tight interaction of new cloud technologies with SOA technologies enables us to optimize the execution of data-intensive service applications by reducing the data exchange tasks to a minimum. Fundamentally, our core concept to optimize the data flows is found in data clouds. Moreover, we can exploit our approach to derive efficient process execution strategies regarding different optimization objectives for the data flows.**

***Keywords*-service applications; data-intensive; data cloud;**

## I. Introduction

The service-oriented architecture is a widely accepted and engaged paradigm for the realization of business processes that incorporate several distributed, loosely coupled partners. Today, Web services and the Business Process Execution Language for Web Services (BPEL4WS, BPEL for short) [1], [2] are the established technologies to implement such a service-oriented architecture [1], [2]. The functionality provided by business applications is enclosed within Web service software components. Those Web services can be invoked by application programs or by other Web services via Internet without explicitly binding them. On top of that, BPEL has been established as the de-facto standard for implementing business processes based on Web services [1], [2].

Aside from business processes, the service-oriented approach using Web services and BPEL is also of great interest for the implementation of data-intensive processes such as, for example, those found in the area of data analytics. Generally, we define the notion of a data-intensive process as a collection of related structured activities or tasks (services) that produce a specific result; huge data sets have to be exchanged between several loosely coupled services in such a data-intensive process. In this case, the exchange of massive data is challenging and several papers have shown that the preferred XML-based SOAP protocol [2] for the communication between Web services is not efficient enough in those scenario settings [3], [4], [5], [6].

To tackle this data exchange issue on a conceptual level, we proposed specific extensions on the Web service [4] as well as the BPEL levels [7] for data-intensive service applications. Using these extended technologies, the exchange of large data sets between participating services in a process is conducted with specific data-oriented approaches, such as ETL tools for the data exchange between databases [8]. The incorporation of such specific data propagation tools is done in a transparent way using service technologies during the process execution. The main disadvantages of our approach can be characterized as follows:

1) The specific data propagation/exchange services have to be available at process execution time. If this demand is not met, the process execution is not possible; this clearly restricts the practicability and applicability of our approach. Therefore, the overall approach depends on the availability of such specialized data propagation services.
2) The performance of the data exchange is dramatically improved by this approach, as proven in [7]. However, the data exchange is still a time-consuming activity, which should not be underestimated. Therefore, further improvements have to be done to optimize the overall performance of data-intensive service applications.

Up to now, our proposed conceptual design perfectly fits our main realistic assumption: *each loosely coupled service has its own storage system*, e.g., a database system. This storage system is utilized to efficiently process large amounts of data within the service. However, based on work in the direction of cloud technologies, the assumption can be refined in a novel way. Fundamentally, the cloud is a metaphor for the Internet, and it is an abstraction for the complex infrastructure it conceals. Therefore, a data cloud is a widely available abstract data layer offering scalable, reliable, cheap and speedy access to data over the Internet. Examples of a data cloud are *Amazon SimpleDB* [9] or

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

*HadoopDB* [10]. According to this, we define our main assumption as follows: *Each loosely coupled service uses the same data cloud as common, shared storage and data processing system.*

Using our so-called *data cloud assumption*, we are able to simplify the execution of data-intensive service applications in two ways: (i) specific data propagation services do not have to be available at process execution time, and (ii) the data exchange between participating services in a process is completely replaced by the exchange of data references as pointers. Based on this issue, the overhead of the data propagation—by value exchange—is reduced to a minimum. Fundamentally, we only want to propagate data in those situations where we cannot avoid it at all.

In order to reach our described properties, we review the proposed underlying infrastructure [4], [7] in detail in Section II. Then, we introduce our specialized variant of *Data Cloud Web Services* as an extension of [4], which incorporates our main assumption in Section III. While Section III focuses on the service level, Section IV introduces all concepts for the process modeling and execution level. Section V presents our data cloud abstraction approach to overcome some of the limitations introduced earlier. Moreover, we present several strategies to initialize process executions regarding different optimization objectives in this section. Finally, we conclude the paper with a summary and an outlook in Section VII.

## II. Preliminaries

As already mentioned, the service-oriented approach should be efficiently applicable for other domains, such as data-intensive service applications, e.g., for data analysis processes [11]. As emphasized several times in [3], [4], [7], [5], [6], Web services as well as BPEL suffer from certain shortcomings regarding the exchange of large amounts of data between Web services. To tackle these shortcomings, extensions on the service level [4] and on the process level [7] have been proposed. In this section, we review these extensions and present their main disadvantages.

### A. Service-Level Extension

The concept of *Data-Grey-Box Web Services (DGB-WS)* is a specific extension of the Web service technology for data-intensive service applications [4]. Each DGB-WS exhibits its own storage system—e.g., a database system as illustrated in Figure 1—that can be used to efficiently process large amounts of data within the service. However, in contrast to the original black-box Web service approach [2], the Web service interface of the Data-Grey-Box Web services is enhanced with an explicit data aspect offering exhaustive information about the data semantics. Aside from the separation of functional parameters and data in the interface description, a novel binding format for structured data was introduced. Through this new data binding, services signal

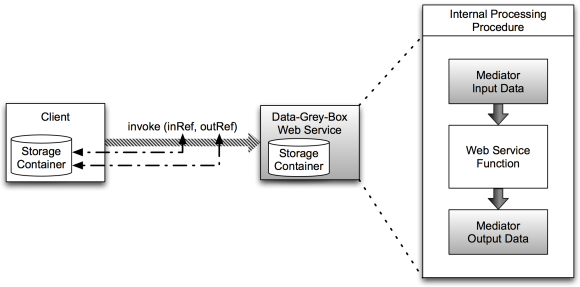


Figure 1. Operation Method of Data-Grey-Box Web Services.

that data has not been transferred via SOAP and that there is a separate data layer instead. As before, regular functional parameters are handed over via SOAP when calling the Web service.

To handle this newly introduced data binding, the SOAP framework was extended with the integration of a novel data layer component. On the client side, enhanced Web service call semantics are necessary. Besides the transmission of the endpoint information and regular parameters in the SOAP message, the client has to deliver access information as references for (i) where the input data is available (input reference) and (ii) where the output data should be stored (output reference). Thus, the new data binding is translated into no more than two additional parameters with access information for input and output data on the client side. These new parameters are included in the SOAP message for the invocation of Web services. The advantage of this procedure is that instead of propagating the pure data in an XML-marshaled SOAP message, only the access information in the form of data pointers are delivered in SOAP. This property is depicted in Figure 1, where the client delivers data pointers to its storage system to the service.

On the service side, the extended SOAP framework receives the SOAP message and conducts a separation into the functional aspect and the data aspect. The associated data layer calls an appropriate mediator for the data propagation based on the access information of the client and the service. While the client's data access information can be found in the received SOAP message, the data access information for the service instance must be queried from the extended service infrastructure [4]. Fundamentally, a mediator is a neutral third party member that is responsible for the data propagation between client and Web service. Such mediators have to be accessible via a standard Web service interface. Using this mediator approach, the heterogeneity of the storage system is tackled on a conceptual level.

If a DGB-WS receives and returns a data set as illustrated in Figure 1, two data propagation tasks via mediators at the service side will be initiated. The first propagation task for the input data is conducted before the pure functionality of the service is invoked. The correlation between such input

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

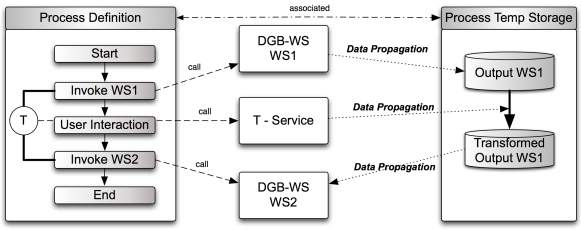


Figure 2. BPEL$^{DT}$ - Modeling and Execution Perspective.

data and the Web service instance is realized by our extended service infrastructure. If the input data propagation task is finished, the functionality is automatically invoked. The last step is the initiation of the data propagation task to deliver the output data to the client.

Fundamentally, this proposed concept of *Data-Grey-Box Web services* offers several drawbacks. One drawback is that the client has to deliver access information to its internal storage system. Another drawback is the restricted usability of Data-Grey-Box Web services, which is linked to the availability of appropriate mediator services to propagate data between the participating storage systems. If no appropriate mediator service is available, either the client cannot invoke the service or the client has to change its own storage system. The data exchange will also be initiated when the storage systems of client and service are equal.

### B. Process-Level Extension

In order to efficiently realize comprehensive data-intensive service applications, the next step is the orchestration of Data-Grey-Box Web services. Therefore, BPEL data transitions, as a data-aware extension of BPEL, have been proposed in [7]. These data transitions are explicit link types connecting several Data-Grey-Box services on the data level. Fundamentally, those data transitions are an orthogonal data flow concept compared to the control flow.

In the left part of Figure 2, a simple process consisting of two service invocations (DGB-WS), $WS1$ and $WS2$, with a user interaction between these invocations, is illustrated, where the user's input is necessary to call service $WS2$. Furthermore, both services, $WS1$ and $WS2$, are also explicitly connected on the data level with a data transition (illustrated by a solid line). The meaning of the data transition is that the output data of $WS1$ is used as input data for $WS2$. Moreover, the data transition includes a schema transformation specification to construct the input data for $WS2$ according to the necessary schema from the output data of $WS1$ (illustrated by a circle containing the character $T$). As a consequence, data transitions include a specification of how the output data of a source service has to be transformed to the input data schema of the target service.

Aside from the process definition, Figure 2 illustrates the process execution on an abstract level as well (using dashed lines). An adapted BPEL engine also has an explicit storage system as a temporary storage location. As depicted, the output data of $WS1$ is stored at this position. Then, the schema transformation of the data transition is executed (*T-Service*) with the help of the temporary storage system. Afterwards, Web service $WS2$ gets its input data from this storage system. In summa, three explicit data propagation (by value) tasks using specialized propagation tools are conducted during the process execution. In [7], we have shown that such an execution strategy performs better than the traditional SOAP-based approach.

### C. Cloud Technologies

The term cloud computing points to the following aspects: (i) applications delivered as services over the Internet and (ii) the hardware and system software in the data centers that provide those services. As already mentioned by Anstett et al. [12], three different delivery models have been introduced in the area of cloud computing over the last years: (i) *infrastructure as a service (IaaS)*, (ii) *platform as a service (PaaS)* and (iii) *software as a service (SaaS)*. These three models are exhaustively described in [12].

In this paper, we focus on infrastructure as a service, which provides the basic infrastructure to the customer. This infrastructure model can be distinguished into storage (data cloud) and computing capabilities (resource cloud). The advantage of this model is that the complexity of infrastructures is completely concealed from developers and users. Thus, developers as well as users do neither know nor require to know what is in the cloud; they only care that it delivers the services they need. In the next section, we show how to use a data cloud to optimize the data-intensive service applications.

## III. Data Cloud Web Services

With the increasing availability and popularity of data clouds, their efficient usage is a challenging task for researchers as well as practitioners. The goal of this paper is to tightly incorporate data clouds into functionally driven SOA technologies to optimize data-intensive service applications. Therefore, we first introduce our concept of *Data Cloud Web services (DC services)* as the foundation for this optimization. Our concept is a specialized variant of Data-Grey-Box Web services, but we start with a new main assumption. As illustrated in Figure 3, every DC service works in combination with a commonly shared data cloud as storage system for the efficient processing of massive data. In this scenario, we see a data cloud as a widely open, scalable and accessible data layer with a well-defined interface for efficient data access. In contrast to the concept of Data-Grey-Box Web services, this is a main shift that allows more flexibility, as we will describe later.

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

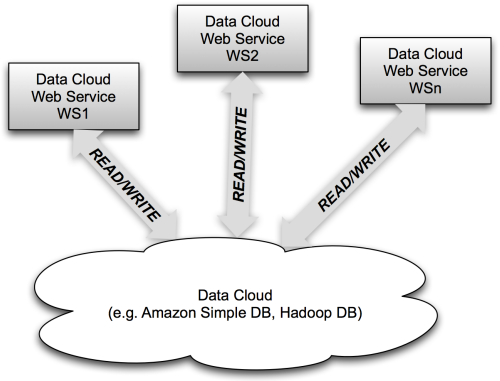


Figure 3. Our Data Cloud Assumption.

Based on the foundation of Data-Grey-Box Web services, DC services have the same enhanced interface description with a separation of functional and data aspects. In this case, the already introduced new *data binding* format signals that data has not been transferred via SOAP and that there is a separate data layer instead [4]. According to the use of the data binding format by Data-Grey-Box Web services, Figure 4 illustrates the adjusted mapping for DC services. Again, on the client side, enhanced service invocation semantics are essential. In such invocation semantics, the endpoint information, regular parameters, and data references are delivered in a SOAP message to the DC service. As depicted in Figure 4, data references are pointers into the data cloud for (i) where the input data is available and (ii) where the output data should be stored. As we can see, there are no differences in the invocation semantics between Data Cloud Web services and Data-Grey-Box Web services on the client side.

However, in contrast to Data-Grey-Box Web services, the internal operation method on the service side is completely different. This difference is clearly noticeable when we compare Figures 1 and 4. Using our main assumption of a commonly shared data cloud for all DC services, we do not have to propagate data from the client storage system to the service storage system. Therefore, we are able to remove the invocation of appropriate mediators from the internal service processing procedure, and the delivered data references from the client are directly passed to functional aspects of the service as main data pointers. These main working data pointers are used to read the input data (READ operation) and to store the output data at the corresponding location in the data cloud (WRITE operation). As already illustrated in Figure 3, the interactions between DC services and the data cloud are restricted to the operations READ and WRITE, and the delivered data references of the client are used to instantiate these operations in an appropriate way. At the moment, these operations are abstract and should

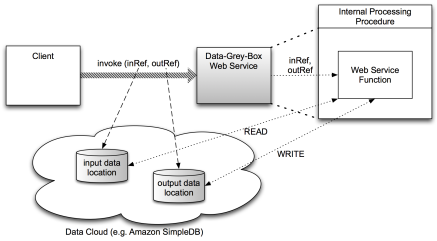


Figure 4. Operation Method of Data Cloud Web Services.

allow efficient access to data in the cloud. Thus, we prefer an SQL-like interface as offered by Amazon SimpleDB [9] or HadoopDB [10] at the moment. Aside from efficient descriptive access to data, such an interface offers further data processing capabilities like grouping or aggregation of data as well. However, this introduce some limitations, which have to addressed in further research activities.

In contrast to the concept of Data-Grey-Box Web services, the internal service processing procedure is simplified. The main advantage of our DC service concept is that the data propagation itself is completely eliminated by the use of a commonly shared data cloud. Instead of exchanging data in a by-value manner—either using SOAP messages or with specialized mediators—, DC services exchange data in a pure by-reference manner, and the internal service procedure guarantees the correct data processing. For data-intensive service applications, this makes a lot of sense and strongly reduces the execution time of services. Another advantage of DC services is the clear definition of the requirements for the invocation. If a client wants to invoke a DC service, the client has to ensure that (i) the input data is available in the data cloud and/or (ii) a storage position for output data is available in the data cloud. In comparison to Data-Grey-Box Web services, these requirements are more strict and the degrees of freedom are fewer.

To summarize, Data Cloud Web services are a specialized variant of Data-Grey-Box Web services. The tight coupling with a data cloud is an outstanding property and enables the elimination of all data propagation tasks within the service invocation. In this case, the service invocation corresponds to a pure functional invocation without any data delivery. If the essential input data is available in the data cloud, DC services are able to access this data directly within the data cloud. Furthermore, the output data of DC services is also put in the data cloud. In the following section, we show how to efficiently exploit these aspects to process data-intensive service applications without the execution of any explicit data propagation task.

## IV. Composition and Execution

In order to create comprehensive data-intensive service applications using several DC services, we employ the BPEL$^{DT}$ approach [7]. From the modeling perspective, we

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

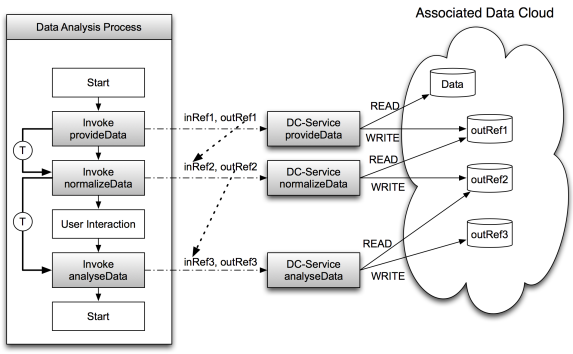


Figure 5. Data-Cloud-Aware Process Execution.

do not have to modify anything in the already proposed approach. The left part of Figure 5 illustrates a sample data analysis process consisting of the following steps:

1) The first DC service *provideData*—after starting the process—provides the data to be analyzed by the subsequent services.
2) The second DC service *normalizeData* conducts a normalization of data. The first and the second DC services are connected by a control link as well as by a data link. The data link—BPEL data transition—signals a data exchange between these services. This data link also includes a specification of how the output data of the DC service *provideData* has to be transformed to the input data schema of the DC service *normalizeData*.
3) The third step is a user interaction to get essential analytics parameters from the user for the subsequent step.
4) The fourth step invokes a DC service *analyseData*, which is responsible for determining the analytic result of the normalized data. Therefore, the DC service *normalizeData* and this DC service are connected by a data link including a schema mapping specification. Afterwards, the process finishes.

In this scenario setting, we assume that the relevant data to be analyzed is already available in the data cloud and is made accessible by the *provideData* DC service. The process execution is depicted in detail in the right part of Figure 5. As in the original BPEL$^{DT}$ approach [7], the process engine is responsible for the data reference handling during process execution. Thus, the process engine is tightly coupled to the same data cloud as the participating DC services. The process execution can be described as follows:

- Based on the functionality of the DC service *provideData*, the data is copied to an engine-determined location $outRef1$ in the data cloud. Those reference information is delivered to the DC service during the service invocation. If the DC service has finished (synchronous invocation), the next process tasks can be executed.
- The engine-determined output data reference of the DC service *provideData* is then used as input data reference for the subsequent DC service *normalizeData* ($outRef1$ equals $inRef2$). The DC service *normalizeData* reads the corresponding data from this location within the data cloud and writes the normalized data to an engine-determined location $outRef2$ using our defined READ/WRITE interface.
- Afterwards, the user interaction is processed and the DC service *analyseData* will be invoked. During this service invocation, the delivered output data reference $outRef2$ of *normalizeData* is used as input data reference $inRef3$. This DC service reads and writes the data according to the engine-determined data cloud locations.

To determine the data reference equalities of input and output data for several DC services, the explicitly available data transitions within the process definition are highly practicable and usable in this case. As we can observe by this example, the process execution is conducted without any explicit data propagation between participating DC services. Instead of propagating data between heterogeneous storage systems, as proposed in [4], [7], we efficiently employ the developed techniques of data clouds and propagate only data cloud reference information between services. Each DC service then operates directly on the delivered data reference locations. Up to now, the described process execution has been simplified by removing the execution of possible data transformation tasks specified within the data transitions. However, we are able to include such tasks easily. One possible way is to invoke a special transformation DC service at any time to execute the delivered transformation. From our point of view, such a functionality should be offered in a data cloud approach.

With our proposed concept of DC services including the adjusted BPEL$^{DT}$ process execution, the data flows within data-intensive service applications are optimized. Instead of propagating massive data sets between heterogeneous systems, we suggest handling all data with a data cloud like Amazon SimpleDB [9], without losing the property of distributed services. However, the tight coupling of all services with one commonly shared data cloud comes with some drawbacks. In the next section, we propose a completely alternative approach to the described concept of Habich et al. [4], [7] to cope with several heterogeneous data clouds.

## V. Data Cloud Abstraction

Our main assumption of a single available and accessible data cloud for all DC services is only valid in a small and restricted real-world environment. Today, we find more than one data cloud, and this eco-system will grow over time. Each of these data clouds can be used as a commonly shared

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

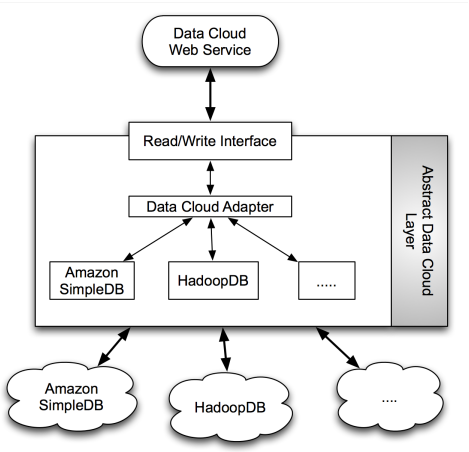


Figure 6. Abstract Data Cloud4 Layer Approach.

storage and data processing system for our DC services. However, if we were to restrict a DC service to a specific data cloud implementation, we would limit this service's longevity in a fast-changing eco-system. To overcome this limitation and to establish more flexibility, we propose an alternative way for the interaction of DC services with several heterogeneous data clouds. One possible solution would be the usage of the Data-Grey-Box Web service concept to tackle the heterogeneity. However, this solution is too inflexible regarding the runtime exchange of the underlying data cloud of a service. Therefore, we propose an alternative approach offering more optimization possibilities, which will be described next.

In order to abstract from a concrete data cloud implementation, we introduce a novel generic data cloud layer called *Abstract Data Cloud Layer*. As depicted in Figure 6, our novel generic data cloud layer is on top of a set of concrete data cloud implementations and offers all operations—READ and WRITE—that are essential for DC services. Aside from providing the appropriate operations, the abstract data cloud layer is also responsible for mapping the operations on different heterogeneous data clouds. For each data cloud implementation, a new adapter can be added to the layer to support this data cloud. This adapter only has to implement our generic *data cloud adapter* interface with the READ and WRITE operations.

Fundamentally, the implementation of a DC service has to be conducted by the use of our new abstract data cloud layer. In this case, each DC service is independent of any concrete data cloud, and this property offers more flexibility regarding the service execution. However, we have to distinguish two strategies—*compile-time* and *runtime* binding—for data cloud bindings, which restricts the execution. On the one hand, the data cloud binding can be defined at the service's compile time. In this case, the developer specifies a binding to a concrete data cloud at deployment time for a DC service. This strategy is essential for the implementation of services that provide e.g. stored data in a specific data cloud on the service level. Another occasion for compile-time binding is the use of specific offered functionalities of a data cloud. On the other hand, the data cloud binding can be defined at runtime, which is our second possible strategy. With this strategy, the business logic is completely compiled to the generic data layer, and the data layer itself can be instantiated at runtime. This runtime configuration can be changed by each single service execution, which allows the possibility to adjust the execution to different environmental settings.

Based on these two data cloud binding strategies–*compile-time* and *runtime* binding—, we are able to create comprehensive processes with the following property: participating DC services in a process are either strictly connected with a data cloud or completely adjustable to different clouds. This special property is illustrated with a small example with two different data clouds in Figure 7. In this example, DC service $WS1$ is linked to $DataCloud1$, $WS3$ is connected with $DataCloud2$, and the data cloud binding for all other DC services can be configured at runtime. The arrows between the DC services indicate the data flows on a high level. To execute this process, we need to specify the data bindings for DC services $WS2$ and $WS3$.

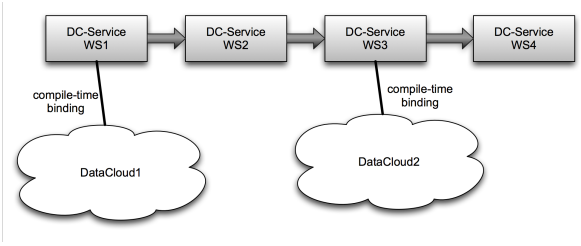


Figure 7. Example Process with a Possible Binding Situation for DC Services.

At a first glance, this additional specification step for the data cloud bindings complicates the process execution. However, from an optimization perspective, this novel specification step enables several possibilities to adjust the process execution regarding different objectives. One objective could be, for example, to reduce the overall cost for the process execution. In this case, all services with a runtime binding opportunity get a binding with the data cloud that produces the lowest costs.

Depending on the optimization objective, we are able to distinguish two strategies for the specification step: (i) *static specification* and (ii) *dynamic specification*. The previous cost objective can be easily realized by a static specification under the same circumstances. The optimization objective of performance is surely an example for a dynamic specification, with the opportunity to react to different workload

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

scenarios in the data clouds. In this case, an autonomous advisor could check the current load situation in the different available data clouds. If the service is invoked, the advisor initializes the data layer with the best data cloud instance available for the service. This enables the whole system to ship the data to the service as fast as possible. The advisor could also use other quality-of-service parameters, not only the workload situation; however, this and the strategy as a whole are only possible if the runtime environment allows such kind of configuration and workload observation.

The detailed technical preparation of the specification step regarding several optimization objectives is our next research topic. The focus of this paper was to present the overall concept. The specification of the data cloud binding for a number of DC services is one necessary step in the direction of process execution, but a further step has to be made. In this next step, data mediator services have to be integrated in the process definition when data cloud changes between consecutive DC services take place. Again, those data mediator services are responsible for the data propagation between several heterogeneous data clouds. To determine the position within the process definition, we use the explicitly available BPEL data transitions and the derived binding specification. Based on the specification strategy, the positions of the data mediators within the process could be determined either statically or dynamically. A possible executable process for our depicted example process of Figure 7 is illustrated in Figure 8. As we can see, $DataCloud2$ is utilized as the main data storage system for all DC services except for $WS1$. Furthermore, after $WS1$, a data mediator service is interlaced in the process definition to propagate the output data of $WS1$ in the data cloud $DataCloud2$. In this case, we assume that a corresponding data mediator is available and configurable to execute the desired task.

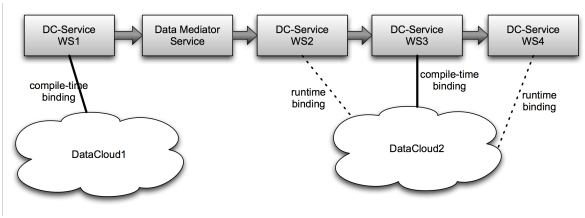


Figure 8. Executable Process Definition for Example Process from Figure 7.

If we compare this approach with the pursued concept of Data-Grey-Box Web services [4], several similarities and differences are observable. In order to tackle the heterogeneity of storage systems, Data-Grey-Box Web services use explicit data propagation tasks with a mediator technique at all points. We describe their approach as data shipping, because the data is always propagated to the function. With our proposed approach in this paper, we are able to ship the function to the data by configuring the functional aspect

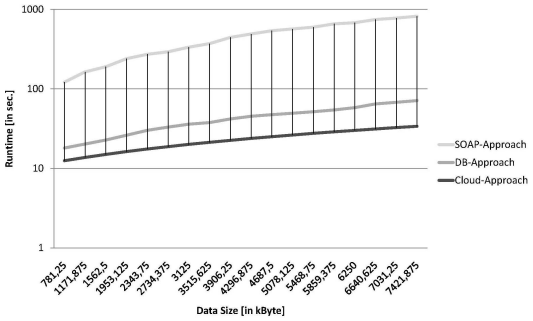


Figure 9. Process Execution Runtimes.

(function shipping). However, if this function shipping is impossible due to certain circumstances, we have to propagate the data to the function as well. In those situations, we adopt the proposed mediator technique of Data-Grey-Box Web services, where these mediators are interlaced in the process definition. Based on this function shipping property, we are able to derive physically executable processes from user-defined process definitions. These executable processes can be optimized regarding different objectives. From our point of view, this novel capability for the process domain offers a lot of research opportunities.

## VI. Evaluation

According to the conducted evaluation in [4] and [7], we have chosen a similar evaluation strategy. We utilize as evaluation process our sample data analysis scenario from Section IV. In this experiment, we measured the runtime of the whole processes by varying the data size to be processed. The results are illustrated in Figure 9, whereas we measured the execution runtime for (i) the classical process execution using SOAP messages (*SOAP approach*), (ii) BPEL$^{DT}$ execution where each Data-Grey-Box Web services has its own storage system (*DB-Approach*), and (iii) our new cloud-aware execution using one single data cloud for all Web services (*Cloud-Approach*). In all experiments, each participating service in the process ran on its own server. For the Data-Grey-Box Web service approach, an open-source database system was installed on the same service server. In the case of our cloud execution, we utilized a HadoopDB [10] instance as central storage system for all services. As we can see in Figure 9, our new cloud-aware process execution outperforms the rest. In contrast to the BPEL$^{DT}$ approach, we save the whole time for the explicit data propagation.

## VII. Summary and Outlook

In this paper, we presented an approach to optimize data-intensive service applications using data cloud technologies. Examples of data-intensive service applications are data analytics processes, which can be found in various domains. The optimization of such processes is always an ongoing research

Final edited form was published in "IEEE 3rd International Conference on Cloud Computing. Miami 2010", S. 19-26, ISBN 978-1-4244-8207-8
https://doi.org/10.1109/CLOUD.2010.56

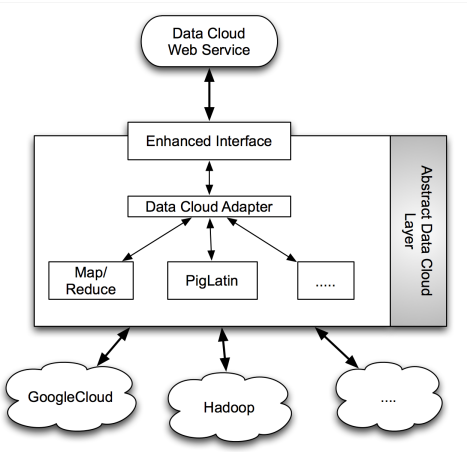


Figure 10. Enhanced Abstract Data Cloud Layer Approach.

topic. Our proposed solution consists of an enhanced Web service concept called *Data Cloud Web services*, and each Data Cloud Web service works in tight combination with a commonly shared data cloud as storage and data processing unit. Based on this property, we are able to replace every explicit data propagation with exchanged data references as pointers in the cloud. As a second step, we described the execution of comprehensive process logics. Third, we proposed an approach to abstract from concrete cloud implementations that offers more flexibility regarding the service and process execution. Our abstraction approach enables us to ship functions to the data instead of shipping data to the functions, as it is the preferred standard method in service-oriented environments. Our complete proposed approach is realized within our Open Service Process Platform 2.0 [13], whereas we are able to utilize the Amazon SimpleDB [9] or HadoopDB [10] as data clouds for our approach.

Our next research topics are manifold. One research direction focuses on the detailed technical preparation of the specification step regarding several optimization objectives as described in Section V. However, we consider these works only as one step in the right direction to efficiently utilize the full power of cloud technologies in combination with SOA technologies. A major drawback of our current approach is our limited READ-WRITE interface to access heterogeneous data clouds. Today, several developed data processing standards on top of specialized data clouds exist. The most common ones are *Map/Reduce* [14] and *PigLatin* [15] with several assets and drawbacks. In further research work, we are going to enhance our *Abstract Data Cloud Layer* with those standards. As depicted in Figure 10, DC services are implemented against our abstract enhanced interface, and our cloud layer is able to derive different concrete cloud-aware implementations. Thus, our overall approach is no longer restricted to the data handling, and we include the efficient data processing in our consideration. Then, we are able to ship the function to the data and we can adjust the implementation of the function.

## REFERENCES

[1] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR, 2005.

[2] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson., *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, 2005.

[3] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the limits of soap performance for scientific computing." in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 246–254.

[4] D. Habich, S. Preißler, W. Lehner, S. Richly, U. Aßmann, M. Grasselt, and A. Maier, "Data-grey-box web services in data-centric environments," in *Proc. of ICWS*, 2007, pp. 976–983.

[5] A. Ng, "Optimising web services performance with table driven xml," in *Proc. of ASWEC*, 2006.

[6] R. van Engelen, "Pushing the soap envelope with web services for scientific computing." in *Proceedings of the International Conference on Web Services*, 2003, pp. 346–352.

[7] D. Habich, S. Richly, M. Grasselt, S. Preißler, W. Lehner, and A. Maier, "Bpel$^{DT}$ - data-aware extension of bpel to support data-intensive service applications." in *Proc. of WEWST*, 2007, pp. 111–128.

[8] A. Simitsis, "Modeling and managing etl processes," in *Pro. of PhD-VLDB*, 2003.

[9] A. W. Services, "Amazon SimpleDB," http://aws.amazon.com/simpledb/, 2009.

[10] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz, "Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads," *PVLDB*, vol. 2, no. 1, pp. 922–933, 2009.

[11] D. Habich, T. Wächter, W. Lehner, and C. Pilarsky, "Two-phase clustering strategy for gene expression data sets," in *Proc. of SAC*, 2006, pp. 145–150.

[12] T. Anstett, F. Leymann, R. Mietzner, and S. Strauch, "Towards bpel in the cloud: Exploiting different delivery models for the execution of business processes," in *Proc. of IWCS*, 2009, pp. 670–677.

[13] D. Habich, S. Richly, A. Ruempel, W. Buecke, and S. Preissler, "Open service process platform 2.0," in *SERVICES I*, 2008, pp. 152–159.

[14] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.

[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proc. of SIGMOD*, 2008, pp. 1099–1110.