Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Optimizing the efficiency of data-intensive Data Mashups using Map-Reduce

Sunayana Sarangi

Course of Study:	Technikpädagogik
Examiner:	Prof. Dr.-Ing habil. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Pascal Hirmer
Commenced:	December 1, 2016
Completed:	June 1, 2017
CR-Classification:	C.2.4, H.3.4

Abstract

In order to derive knowledge and information from data through data processing, data integration and data analysis, a variety of Data Mashup tools have been developed in the past. Data Mashups are pipelines that process and integrate data based on different interconnected operators that realize data operations such as filter, join, extraction, alteration or integration. The overall goal is to integrate data from different sources into a single one. Most of these Mashup tools offer a grahical modeling platform, enabling the users to model the data sources, data operations and the data flow, thus, creating a so called Mashup Plan. This enables non-IT experts to perform data operations without having to deal with their technical details. Further, by allowing easy re-modeling and re-execution of the Mashup Plan, it also allows an iterative and explorative trial-an-error integration to enable real time insights into the data. These existing Data Mashup tools are efficient in executing small size data sets, however, they do not emphasize on the run-time efficiency of the data operations. This work is motivated by the limitations of current Data Mashup approaches with regard to data-intensive operations. The run-time of a data operation majorly varies depending on the size of the input data. Hence, in scenarios where one data operation expects inputs from multiple Data Mashup pipelines, which are executed in parallel, a data intensive operation in one of the Data Mashup pipelines leads to a bottleneck, thereby delaying the entire process. The efficiency of such scenarios can be greatly improved by executing the data-intensive operations in a distributed manner. This master thesis copes with this issue through an efficiency optimization of pipeline operators based on Map-Reduce. The Map-Reduce approach enables distributed processing of data to improve the run-time. Map-Reduce is divided into two main steps: (i) the Map step divides a data set into multiple smaller data sets, on which the data operations can be applied in parallel, and (ii) the Reduce step aggregates the results into one data set. The goal of this thesis is to enable a dynamic decision making while selecting suitable implementations for the data operations. This mechanism should be able to dynamically decide, which pipeline operators should be processed in a distributed manner, such as using a Map-Reduce implementation, and which operators should be processed by existing technologies, such as in-memory processing by Web Services. This decision is important because Map-Reduce itself can lead to a significant overhead while processing small data sets. Once it is decided that an operation should be processed using Map-Reduce, corresponding Map-Reduce jobs are invoked that process the data. This dynamic decision making can be achieved through *WS-Policies*. Web Services use policies to declare in a consistent and standardized manner what they are capable of supporting and which constraints and requirements they impose on their potential requestors. By comparing the capabilities of the Web Service with the requirements of the service requestor, it can be decided if the implementation is suitable for executing the data operation.

Kurzfassung

In den letzten Jahren lässt sich eine zunehmende Entwicklung von Data Mashup-Ansätzen und -Werkzeugen beobachten. Diese bieten einen einfachen und schnellen Weg, um Daten zu verarbeiten und analysieren. Data Mashups bestehen aus Datenquellen sowie einer Reihenfolge von Datenoperationen, wie Filter, Extraktoren usw. und ermöglichen eine Integration der Datenquellen. Dadurch können aus Daten wichtige Informationen und Wissen generiert werden. Diese Werkzeuge bieten meistens eine grafische Oberfläche und ermöglicht dabei eine einfache Bedienbarkeit durch Domänennutzer sowie eine explorative Vorgehensweise. Allerdings legen die vorhandene Ansätze keinen Wert auf die Effizienz der Ausführung. Dies kann daran liegen, dass durch Data Mashups in der Regel kleine Datenmengen verarbeitet werden. In der heutigen Zeit steigt die Datenmenge immer weiter an und die Data Mashup-Ansätze müssen sich anpassen, um die Verarbeitung bzw. Integration von größeren Mengen an Daten zu ermöglichen. Dabei spielt auch die Effizienz der Ausführung eine sehr wichtige Rolle. Bei Data Mashup kann es auch dazu kommen, dass sowohl kleine als auch große Daten gleichzeitig verarbeitet bzw. intiegriert werden müssen. In solchen Fällen führen die daten-intensiven Operationen zum Engpass und der gesamte Prozess muss auf den Engpass warten. Um mit diesem Problem umzugehen muss die Datenverarbeitung, abhängig von Parametern wie Datengröße, Komplexität der Operation bzw. Daten, entsprechend unterschiedlich durchgeführt werden. D.h., es müss unterschiedliche Implementierungen für unterschiedliche Datengröße sowie Komplexität der Operation bzw. Daten geben. Durch solche selektive Verfahren kann die Effizienz des Data Mashups gewährleistet werden. Die Auswahl der Implementierungen muss dynamisch geschehen. In dieser Arbeit wird eine Konzept entwickelt, wodurch die oben genannten Probleme behandelt und eine Optimierung der Ausführung erzielt werden kann. Die daten-intensive Operationen werden anhand einer Map-Reduce Implementierung ausgeführt und die Verarbeitung der kleinen Datenmenge wird durch Web Services im Hauptspeicher durchgeführt. Dieses selektive Verfahren ist wichtig, weil es zu hohem Aufwand kommen kann, wenn kleine Datenmenge durch Map-Reduce verarbeitet werden. Die Map-Reduce ermöglicht eine parallelität der Operation, somit wird die Ausführungsdauer verkürzt. Die vorhandenen Implementierungen verarbeiten die Daten als Ganzes und sind daher ungeeignet größere Datenmenge zu verarbeiten. Dagegen ermöglicht der Map-Reduce-Ansatz eine parallele Verarbeitung, was zu einer Effizienzoptimierung führt. Die dynamische Auswahl der Implementierung wird anhand von *Web Service – Policies* gemacht. WS-Policies beschreiben *was der Web Service kann* und *was er von dem Servicenehmer erwartet*. Anhand eines Vergleiches zwischen den Leistungen des Web Services und den Erwartungen des Service-Konsumenten kann entschieden werden, ob der Web Service sich eignet, um die Operation durchzuführen.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

In today's world, IT applications are highly interconnected, thereby producing and consuming huge amounts of highly complex and heterogeneous data [HRWM15]. The amount as well as the complexity of data is expected to dramatically increase in the coming years [HRWM15]. In order to derive information and insights from Big Data[1] using analysis, visualization or similar other value-adding operations, it is necessary to integrate relevant parts of data in to a common source [HRWM15]. These integration solutions should support ad-hoc and flexible data processing capabilities to cope with the dynamic real time environment. In addition, these solutions should allow an iterative and explorative trial-an-error integration to enable real time insights into the Big Data [HRWM15]. Several Data Mashup platforms have been developed in the past to provide data integration solutions. Data Mashups are defined as pipelines that process and integrate data based on different interconnected operators that realize data operations such as filter, join, extraction, alteration or integration. The overall goal is to integrate data from different sources into a single one. Implementation of complex data processing and data integration solutions pose technical challenges to the domain-specific users, who more often than not, are non-IT experts, thus resulting in higher costs and greater communication effort between domain-specific users and IT experts. To cope with this problem, the domain-users need to be equipped with tools which abstract from technical details of the data operations, allowing them to use their domain specific means to perform these data operations. The existing Data Mashup solutions offer graphical user interfaces enabling the domain users to model data sources, data operations and data flow without having to acquire technical know-how of the data operations. For example, the extended Data Mashup approach, FlexMash[2] (Flexible Data Mashups Based on Pattern-based Model Transformation), developed at the University of Stuttgart, introduces the concept of patterns to abstract from implementation details, thereby enabling non-IT experts to perform data operations without having to acquire technical know-how of the data operations.

FlexMash and other existing Data Mashup approaches however do not emphasize on the efficient execution of the data operations, rather are focused on the usability of the

[1] https://www.ibm.com/big-data/us/en/
[2] https://github.com/hirmerpl/FlexMash

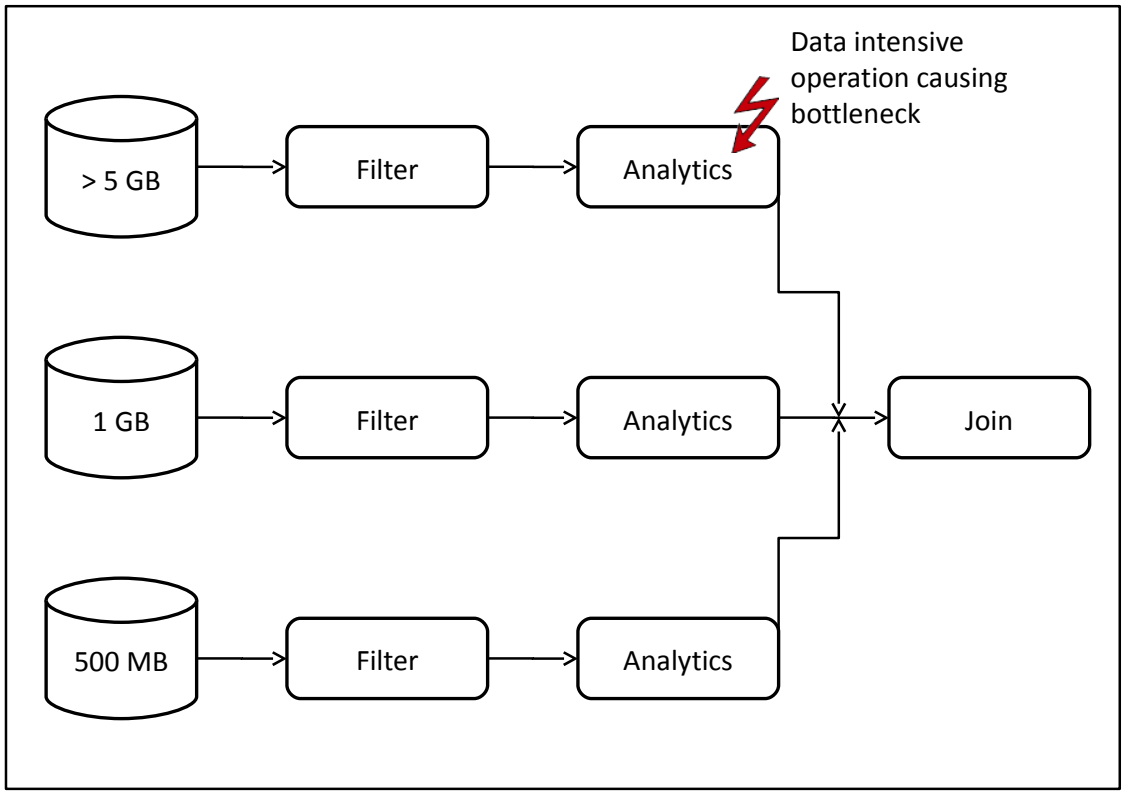

Figure 1.1: Example Scenario

modeling user interface. Data Mashups, until recent times, have mostly been used to process small size data sets and are unable to cope with large volumes of data. With the overwhelming increase in the size and complexity of data, the run-time efficiency of the data operations is becoming a very important factor. Consider the scenario, as depicted in Figure 3.1, consisting of multiple Data Mashup pipelines, each executing data operations on input data sets of varying sizes. Similar input data sets from different sources have to be filtered, analyzed and integrated. The run-time of each pipeline is directly proportional to the size of the input data set. The long running *analytics* node in the upper pipeline acts as a bottleneck. As a result, the join operation to integrate the data has to wait until the slowest path has finished execution, i.e., the overall execution time of Data Mashups will always depend on the slowest path. This work proposes a possible efficiency optimization of pipeline operators using Map-Reduce[3]. The Mashup efficiency will be improved when all the data operations execute efficiently. For this to

[3]http://hadoop.apache.org/

happen, the implementation of a data operation has to take into account the influencing factors, such as data size, data complexity, computing resources etc. Depending on these factors, the modeled data operation has to choose an appropriate implementation for the operation. For example, smaller input data sets can be processed efficiently in-memory by the existing Web Services but for large data sets, executing the data operations in a distributed manner, using technologies such as Map-Reduce, can be more efficient. In FlexMash, this decision making can be achieved by annotating the Web Services using WS-Policies[4]. WS-Policy describes the non-functional constraints, capabilities and requirements of the respective Web Service to which it is attached. Since WS-Policies are a part of the WSDL, an appropriate implementation can be chosen by comparing the offered capabilities/constraints with the factors impacting the run-time efficiency of a data operation.

Structure

The remainder of this paper is structured as follows:

Chapter 2 – Basic Concepts: This section describes the basic concepts that are necessary to comprehend the optimization approach proposed in this work.

Chapter 3 – Related Work: Work related to this project is mentioned and briefly described.

Chapter 4 – Efficiency Optimization of data-intensive Data Mashups: This section describes the main contribution of this paper. The approach for optimizing the efficiency of pipeline operators based on Map-Reduce is explained in detail.

Chapter 5 – Prototypical Implementation and Evaluation: A prototypical implementation of the proposed approach is presented and the results are subsequently evaluated in this section.

Chapter 6 – Summary: The results are summarized and important conclusions are drawn in this section.

Chapter 7 – Future Works: This section focuses on the scope of future work.

[4] https://www.w3.org/TR/ws-policy/

2 Basic Concepts

In this section, we describe the basic concepts that are necessary to comprehend the approach presented in this thesis. These are (i) Data Mashups, (ii) Transformation Patterns, (iii) Web Service Policies and (iv) Map-Reduce Framework.

2.1 Data Mashups

The main purpose of Data Mashups is to enable the composition of disparate sources of information into a new integrated source of information that can be accessed independently of the original data providers. This integrated data source is potentially more valuable for activities such as data analysis. In other words, Data Mashup can be thought of as a **Web-based form of data integration**, since the core practice underlying any Data Mashup is typically integration of data from different sources. There are various architectural patterns available for Data Mashups, namely (i) Point-to-Point Data Mashups as illustrated in Figure 2.1, (ii) Centrally Mediated Data Mashups as illustrated in Figure 2.2 and (iii) Data Mashups with External Data Processing Logic as illustrated in Figure 2.3. In the Point-to-Point Data Mashup architecture, data integration is achieved as a result of a direct interplay of data sources with data processing functions or of one data processing function with another. The Mashup establishes the necessary direct point-to-point communications. The centrally mediated Data Mashups are very similar to Point-to-Point Data Mashups, with the exception that data is transformed and stored in an integrated data source, and all data processing functions operate on this integrated data store only. This architecture has two new components: the *data mediator* and the *integrated data source*. The data mediator mediates between the data models of the data sources and that of the integrated data store. The integrated data store hosts the integrated data for the processing of the Mashup logic. Since there is no data mediation in Point-to-Point Data Mashup architecture, each data processing function has to understand two different data models, that is, the data model of the input data and that of the output data. On the other hand, in the centrally mediated Data Mashup architecture, the data processing functions must only understand the data model of the integrated data store. Although the Data Mashup characteristics are essentially the same in both the above mentioned architectures, in the centrally mediated Data Mashup, there

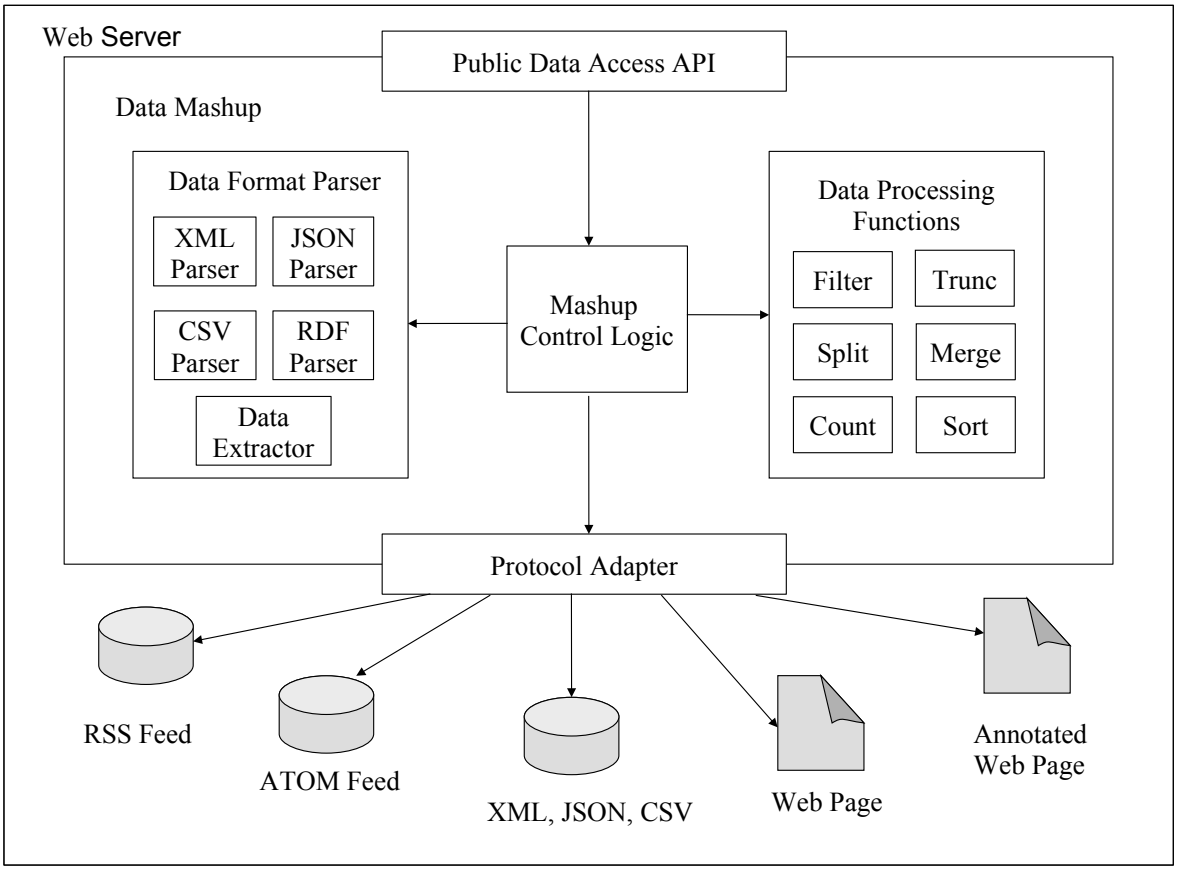

Figure 2.1: Point-to-Point Data Mashup Architecture [DM14]

is no direct data passing between source components and data processing functions or between functions themselves. Data passing between the source components and the integrated data source is mediated and the data passing between the data processing functions is typically based on a shared memory, i.e., the integrated data store. The third architecture, i.e., Data Mashup with external data processing logic, is essentially a centrally mediated Data Mashup architecture. The only difference is that, besides internal data processing functions, it also makes use of Web Services or third-party data processing capabilities [DM14].

2.2 Transformation Patterns

Patterns are used to document proven solutions to recurring problems in a specific context. The concept of patterns, initially introduced in the domain of architecture [AIS78], is now popular in various areas in computer science and information technology.

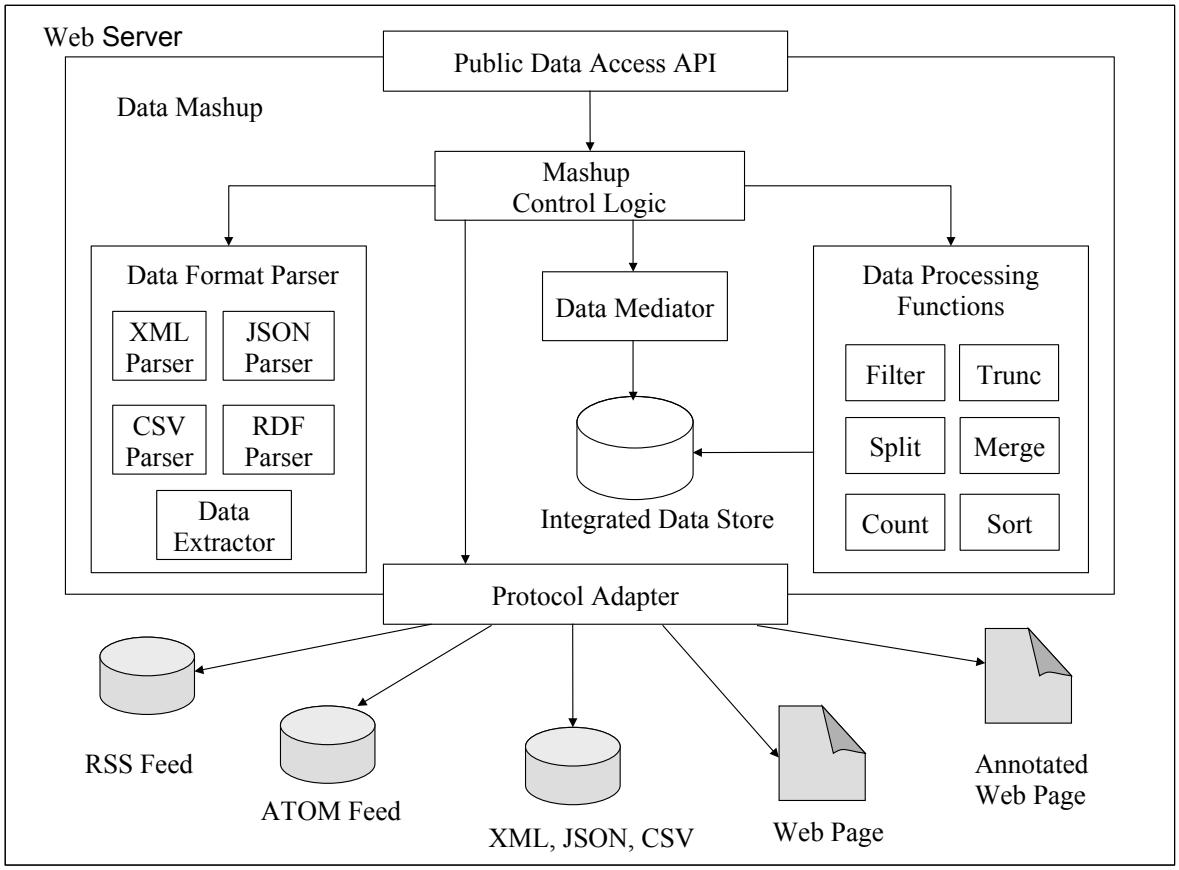

Figure 2.2: Centrally Mediated Data Mashup Architecture [DM14]

Patterns basically document the solutions in natural text to support human readers and are specific to a context [FBB+14]. They provide a generic description of the proven solutions at a conceptual level. To apply this high-level, abstract and conceptual solution to a concrete problem scenario is often difficult, as it involves immense manual effort to concretize the solution to suit the individual use case. This, however, contradicts the very purpose of patterns, since the concept of patterns fundamentally aims at generalization and abstraction. In order to overcome this problem, Falkenthal et al. [FBB+14] suggest that patterns should be linked (i) to the original concrete solutions from which they have been deduced and (ii) to the individual new concrete implementations of the abstractly described solutions. This will enable users to apply a pattern by using the already existing implementations, if any, for their use cases, thus, reducing the manual effort in re-implementing the abstractly described solution.

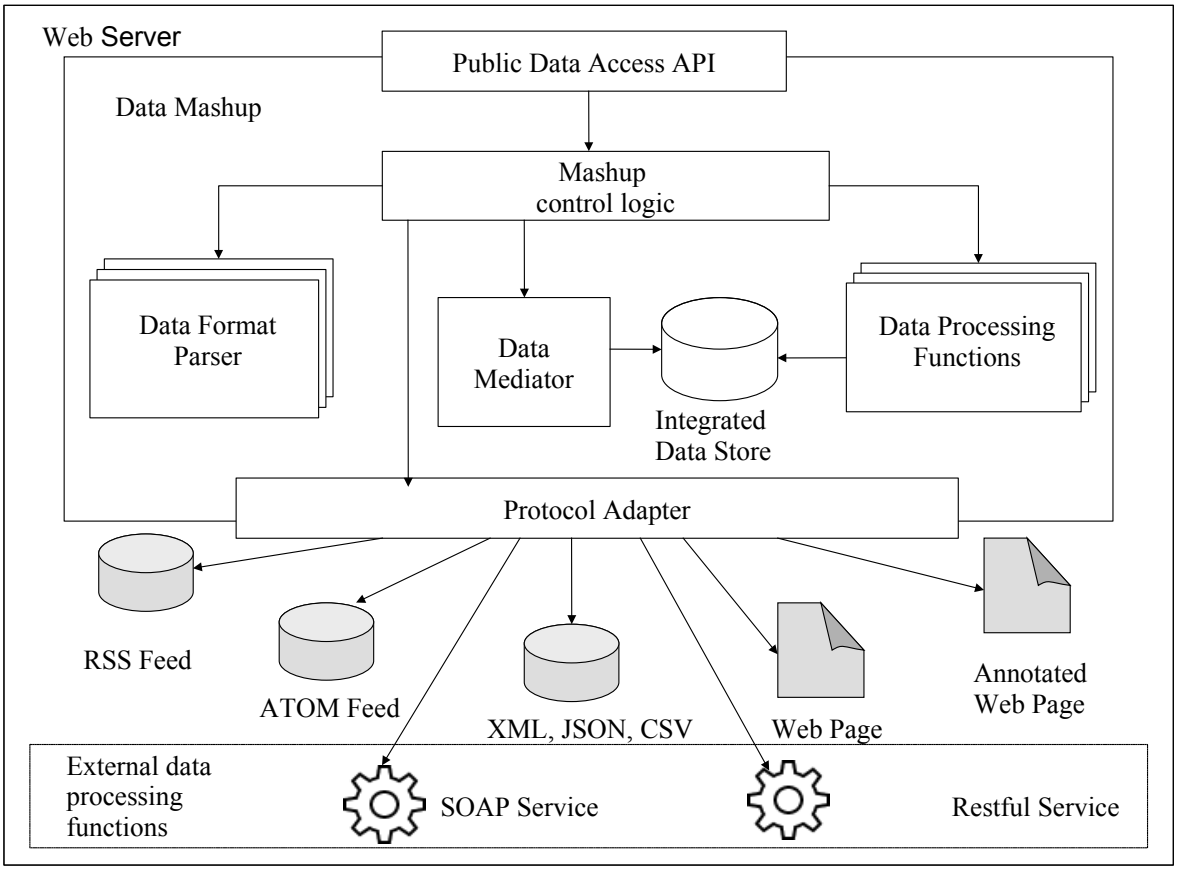

Figure 2.3: Data Mashup with External Data Processing Logic [DM14]

Figure 2.4 and Figure 2.5 illustrate two patterns that are used in Data Mashup scenarios in FlexMash[1], a Data Mashup tool developed at the University of Stuttgart. In the context of FlexMash, these patterns are categorized under *Transformation Patterns* because the non-executable Mashup Plan modeled by a domain user is transformed into an executable Mashup Plan on the basis of such patterns. The pattern determines the manner in which the Mashup should be executed. For example, a *Time-Critical Mashup Pattern* enforces a quick execution of the Data Mashup. It is also possible to combine patterns. But to combine patterns, the user performing the Data Mashup must be aware of the limitations of the patterns. For example, the patterns illustrated in Figure 2.4 and Figure 2.5 cannot be combined in the same use case, since a time-critical implementation cannot be robust at the same time. The illustrated patterns also enable the user to take reasonable decisions regarding how to apply patterns by providing him with relevant information. The pattern, typically describing a single transformation criteria, consists

[1]https://github.com/hirmerpl/FlexMash

Transformation Pattern: Robust Mashup

Problem: Robustness is an **important factor** for IT systems, especially in **enterprise applications** and systems. It stands for many factors such as **stability**, **error tolerance**, **logging** etc. that have to be fulfilled in a robust environment.

Solution: A **robust execution engine** that supports error handling, logging as well as data persistence is used.

Example: An exemplary pattern implementation could, e.g., be realized using a **workflow engine** such as Apache ODE, the Oracle workflow engine or the WSO2 engine using BPEL as execution language. These engines **provide all the necessary factors** to ensure robustness.

Evaluation: The Robust Mashup pattern can be used in enterprise environments in which, e.g., workflows are already established. By using this pattern, **robustness can be guaranteed** which is the most important factor in enterprises. However, of course there are some **setbacks regarding runtime efficiency**.

Combination: Secure Mashup Pattern; Big Data Mashup Pattern;

Figure 2.4: Robust Mashup Pattern [HB17]

Transformation Pattern: Time-Critical Mashup

Problem: The scenario the Mashup is executed in is very time-critical, i.e., it is of vital importance that the Mashup is executed as fast as possible to receive the integrated result. Especially in situation recognition and exception escalation scenarios, time is the most important factor. That is, the time-critical requirement is above all other requirements such as robustness, security, data persistence etc.

Solution: A fast execution engine is chosen with a suitable execution language. The Mashup Plan will be mapped onto this language and execute into the engine.

Example: We use the fast and efficient, however, non-robust execution engine Node-RED which has a lack of robustness regarding data persistence, error handling etc. However, the time-critical requirement is fulfilled.

Evaluation: Using this pattern enables its advantages especially in time-critical scenarios such as the integration of sensor data for situation recognition. However, the disadvantages of this pattern are many. By just including the time-critical requirement, no important factors such as error handling, logging or data persistence can be provided. This can lead to an unstable Mashup execution. In conclusion, this pattern should be used connected with other patterns or in prototypical, non-production environments.

Figure 2.5: Time-Critical Mashup Pattern [HB17]

of the following parts: (i) a description of the problem to which the pattern can be applied, (ii) the solution that the pattern offers, (iii) an example of how the pattern can be applied, (iv) an evaluation of the usability of the pattern, thus, enabling the user to decide if the pattern suits his use case and (v) information about if and how the pattern can be combined with other patterns [HB17]. The pattern selected while modeling the Data Mashup influences the manner in which the Data Mashup is executed. In other words, depending on the pattern selected, a suitable workflow engine and appropriate execution components for the Data Mashup operations are selected to transform the modeled Mashup Plan into an executable one.

2.3 Motivation for WS-Policy

In a service-oriented environment, interoperability between Web Services and a standardized representation of non-functional capabilities and requirements of service endpoints is a very important aspect. When Web Services require requestors to follow a specific behavior or when they implement certain protocols on the service-side that impacts requester requirements or expectations, it is essential to communicate these to the potential requester. Web Service Policies provide an important standard to achieve this interoperability. Web Services are in general described using the Web Service Description Language (WSDL). These service descriptions define the meta-data that describe the functional characteristics of Web Service. This meta-data provides information necessary to deploy and interact with a Web Service. WSDL is an XML vocabulary having two parts: (i) a reusable abstract part describing the operational behavior of Web Services, i.e., what the Web Service does in terms of the messages it consumes and produces and (ii) a concrete part which allows to describe how and where to access a service implementation. WSDL limits itself to describing a few key aspects of a service, such as (i) the message formats, (ii) the message interaction pattern, (iii) the way the messages should be represented and (iv) where those messages should be sent. WSDL however is extensible, thereby allowing the inclusion of additional characteristics such as costs to use the service, security characteristics etc. The author needs to use an appropriate language syntax and insert the description at the right place in the WSDL. This extensibility has enabled WSDL to be as tightly defined as possible. WSDL describes *what* a service can do, however, does not provide information about *how* the service implements the business interface, the permissions or constraints it expects of or provides to service requestors and what is expected or provided in a hosting environment. The inherent extensibility of XML and WSDL can be used to describe this information. However, a better approach is the WS-Policy, an extensible framework that is intended to specifically deal with the definition of constraints and conditions. This framework enables constraints and conditions to be composable and supports valid intersections where multiple options

are available. WS-Policy has many advantages over the extension of WSDL. It facilitates the separation of concerns by avoiding a single monolithic specification, i.e. the WSDL, to deal with all the diversity of service description information. WSDL focuses on the functional descriptions and WS-Policy deals with the non-functional descriptions and quality of service aspects. Further, it may be necessary to add additional capabilities to an existing service without impacting the application. In this way, services can offer different qualities of service to different target audiences. Such an incremental updating of service descriptions is not supported by WSDL and can be achieved through WS-Policies. WS-Policy Attachments provide support to flexibly add policies to preexisting services [WCL+05].

Web Service Policies are particularly relevant in the following scenarios: (i) Development and deployment of service requestors: WS-Policy, together with WSDL, describes any requirements the service requester has to fulfill to be able to invoke and use the service. This guides the development and deployment of the service requester. (ii) Service discovery and selection: WSDL descriptions enable service requestors to locate services which satisfy their functional requirements. Similarly, WS-Policy descriptions can be used to locate services which satisfy the non-functional requirements. (iii) Dynamic update of requester configuration: requester and services can exchange policies using *WS-MetadataExchange* port types. When services update their configuration at run-time, the requestors can consequently reconfigure their run-time by retrieving the updated policy information from the service provider [WCL+05].

The WS-Policy Framework consists of two specifications: (i) WS-Policy and (ii) WS-Policy-Attachment. The *WS-Policy* specification provides a flexible and extensible grammar for expressing the capabilities, constraints and requirements of Web Services as policies. It also describes basic mechanisms for the merging of multiple policies that apply to a common subject and the intersection of policies to determine compatibility. The *WS-PolicyAttachment* specification describes how to associate policies with a subject [WCL+05].

2.4 Map-Reduce Framework

Map-Reduce is a scalable and fault-tolerant data processing approach that enables parallel processing of massive volumes of data using many computing nodes, called a Cluster. The Map-Reduce Framework is a programming model as well as a framework that supports the model. The model hides details of parallel execution thus allowing the users to focus on data processing strategies [LLC+12]. The Map-Reduce model comprises of two primitive functions: *Map()* and *Reduce()* which are functionally based on the *Map()* and *Reduce()* functions of functional programming languages. The logical

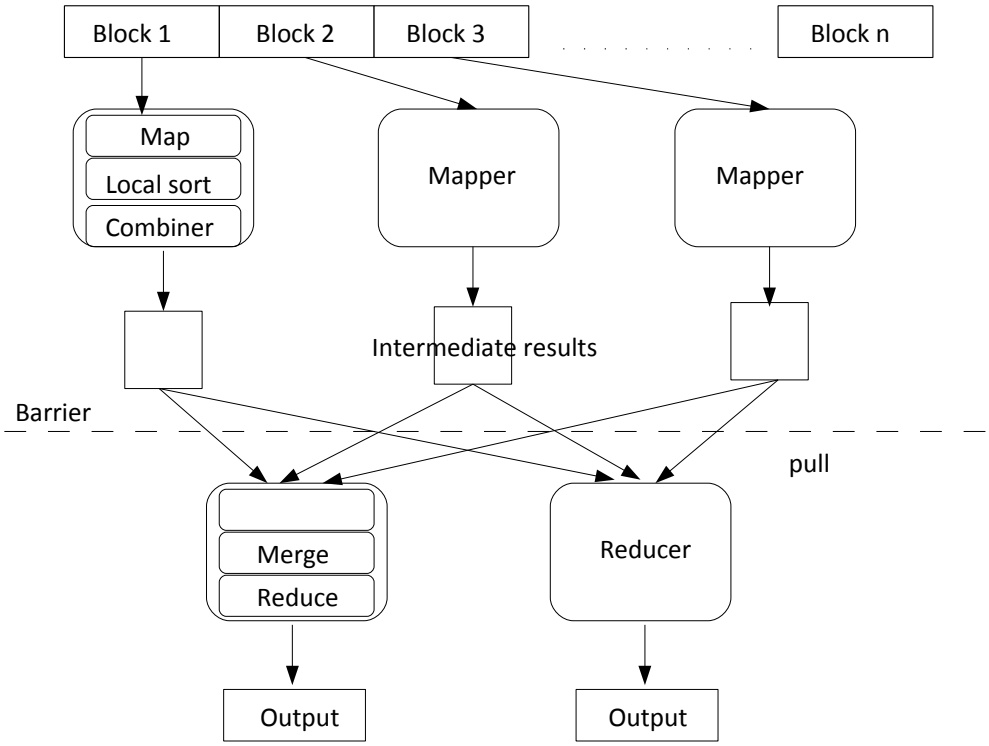

Figure 2.6: Hadoop Architecture [LLC+12]

implementation of the Map and Reduce steps in the Map-Reduce framework is however implementation and use case dependent. What is common between the Map and Reduce steps of the Map-Reduce framework and the map and reduce functions of the functional programming languages is the input and output formats [12]. The Map-Reduce model expects as input a list of *(key1, value1)* pairs and the *Map()* is applied to each pair to compute intermediate key-value pairs, *(key2, value2)*. The intermediate key-value pairs are then grouped together on the key-equality basis, i.e., *(key2, list(value2))*. For each *key2*, *Reduce()* works on the list of all values and produces aggregated values. The *Map()* and *Reduce()* functions are defined by the user depending on their use case [LLC+12].

Apache Hadoop is an opensource implementation of Map-Reduce and is used as the infrastructure for the prototypical implementation in this thesis. Hadoop consists of two layers: a data storage layer called *Hadoop Distributed File System or HDFS* and a data processing layer called *Hadoop Map-Reduce Framework*. HDFS is a block-structured file system managed by a single master node and each processing job in Hadoop is broken down to as many Map tasks as input data blocks and one or more Reduce tasks.

Figure 2.6 shows an overview of Hadoop architecture. A single Map-Reduce job is performed in two phases: Map and Reduce. Before starting the Map task, the input file is loaded on the distributed file system. At loading, the file is partitioned into multiple data blocks which have the same size, typically 64 MB, and each block is triplicated to guarantee fault-tolerance. Each block is then assigned to a mapper. A mapper is a worker node which applies the *map function* to each record in the data block. The intermediate outputs produced by the mappers are *sorted locally* for grouping (key,value) pairs sharing the same key. After the local sort, a *combine function* is optionally applied to perform pre-aggregation on the grouped key-value pairs so that the communication costs incurred to transfer all the intermediate outputs to reducers is minimized. Then the mapped outputs are stored in local disks of the mappers, partitioned into R, where R is the number of reduce tasks in the Map-Reduce job [LLC+12].

When all Map tasks are completed, the Map-Reduce scheduler assigns Reduce tasks to worker nodes. The intermediate results are assigned to reducers via the HTTPS protocol. Since all mapped outputs are already partitioned and stored in local disks, each reducer simply pulls its partition of the mapped outputs from mappers. Basically, each record of the mapped outputs is assigned to only a single reducer by *one-to-one shuffling* strategy. This data transfer is performed by the reducers' pulling intermediate results. A reducer reads the intermediate results and merges them by the intermediate keys, i.e., *key2*, so that all values of the same key are grouped together. This grouping is done by *external merge-sort*. Then each reducer applies the *reduce function* to the intermediate values for each *key2* it encounters. Finally the output of the reducers are stored and triplicated in HDFS [LLC+12]. It is to be noted that the *number of Map tasks* does not depend on the *number of worker nodes*, but the *number of input blocks*. Each block is assigned to a single Map task. However, all Map tasks do not need to be executed simultaneously and neither are Reduce tasks. For example, if an input is broken down into 400 blocks and there are 40 mappers in a cluster, the number of map tasks are 400 and the map tasks are therefore executed through 10 waves of task runs [LLC+12].

The Map-Reduce framework executes its tasks based on a *run-time scheduling scheme*. This means that Map-Reduce does not build any execution plan that specifies which tasks will run on which nodes before execution. The plan for execution is determined entirely at run-time. This enables Map-Reduce to achieve fault-tolerance by detecting failures and reassigning tasks of failed nodes to other healthy nodes in a cluster. Nodes which have completed their tasks are assigned another input block. This scheme naturally achieves load balancing in that faster nodes will process more input chunks and slower nodes less input chunks in the next wave of execution. Furthermore, Map-Reduce scheduler utilizes a speculative and redundant execution. Tasks on straggling nodes are redundantly executed on other idle nodes that have finished their assigned tasks, although tasks are not guaranteed to end earlier on the new assigned nodes than on the straggling nodes. Also Map and Reduce tasks are executed with no communication with other tasks. Thus,

there is no contention arisen by synchronization and no communication costs between tasks during a Map-Reduce job execution [LLC+12].

3 Related Work

In the past, many Data Mashup solutions have been developed to enable ad-hoc processing and integration of data. These enterprise ready solutions offer a graphical modelling tool, enabling the user to define data sources, data operations and the way data is processed. Some known solutions are Yahoo! Pipes[1], Intel MashMaker[2] and the IBM Infosphere Mashuphub[3]. These solutions, however, offer single and static implementations, thereby making themselves unsuitable for real-time scenarios. Today, in real-time business scenarios, aspects like robustness, security, scalability, time-criticality etc. play a very important role and static solutions are not suitable to cope with heterogenous user requirements. In addition, most of the existing data operation and data integration solutions claim to provide abstract, non-technical models for data processing and integration. However, the abstraction provided is often not sufficient for non-IT experts to perform data operations.

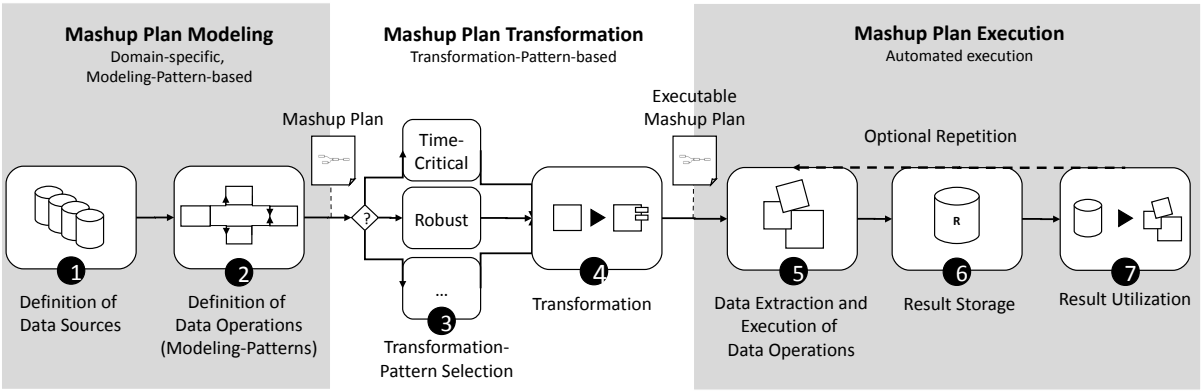

Figure 3.1: Extended Mashup Approach [HRWM15]

The extended Data Mashup approach, FlexMash (Flexible Data Mashups based on Pattern-based Model Transformation), developed at the University of Stuttgart, addresses both these issues. The Data Mashup approach proposed by FlexMash is divided into three abstraction levels: (i) the Mashup Plan Modeling, (ii) the Mashup Plan Transformation

[1]https://pipes.yahoo.com/pipes/
[2]http://intel.ly/1BW2crD
[3]http://ibm.co/1Ghxv27

and (iii) the Mashup Plan Execution, as shown in Figure 3.1. The Mashup Plan Modeling offers a cloud service to the domain user to define and execute Mashup scenarios by exclusively using means specific to his/her domain. A domain-specific model is introduced to define data sources and data operations in a non-technical manner, thus, making it suitable for domain experts.

As shown in Figure 3.2, the domain expert models a non-executable Mashup Plan by defining the order of Data Source Descriptions (DSD) and Data Processing Descriptions (DPD). The Data Source Descriptions describe data sources to be used for the Mashup in a non-technical manner and the Data Processing Descriptions (DPD) are non-technical representations of data operations, for e.g., filter, extract, aggregation etc. These DSDs and DPDs are created and stored in respective repositories by IT experts and the domain users access these repositories and select appropriate DSDs and DPDs to model the Mashup Plan. This enables to abstract from implementation details, thereby enabling non-IT experts to perform data operations without having to acquire technical knowhow of the data operations.

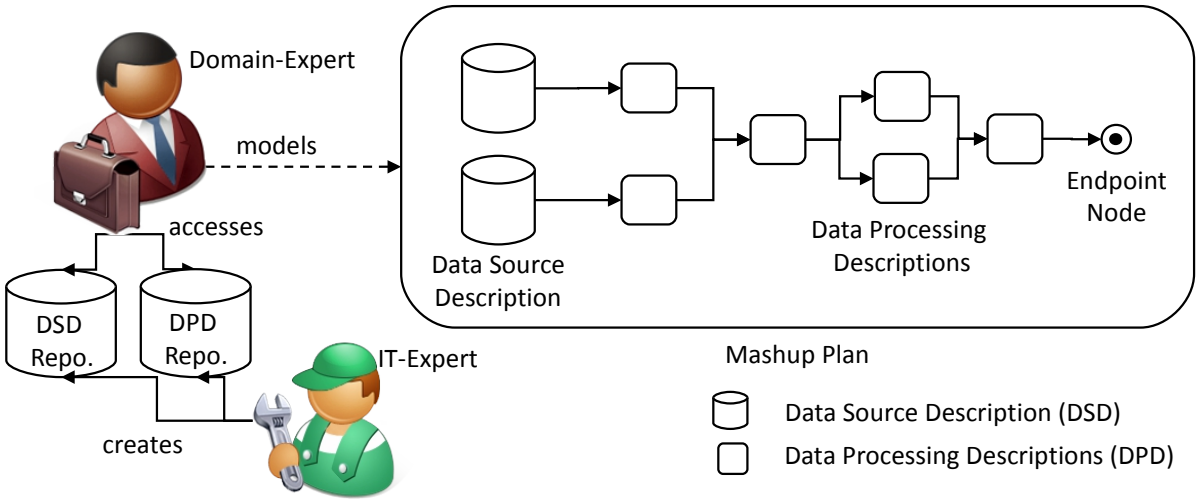

Figure 3.2: Overview of the Modeling Level [HRWM15]

This non-executable Mashup Plan is then transformed into an executable Mashup Plan by applying transformation patterns. The concept of patterns enables to cope with the heterogeneous business requirements. The transformation patterns define criteria under which the execution should be performed, for example, a robust implementation of the Mashup plan or a time-critical implementation. Depending on the transformation pattern(s) annotated to a Mashup Plan, an appropriate implementation, e.g., an appropriate execution engine is chosen. In the prototypical implementation of FlexMash, the *Robust Mashup* is implemented using a BPEL Workflow engine, in order to ensure a high availability and the *Time-Critical Mashup* is implemented using a Node-RED flow. The robust execution has a high run-time due to the heavy-weight workflow

engine that is being used. Additional features such as orchestration, Web Service calls and exception handling lead to a significant overhead. In contrast, the execution of the time-critical Mashup enables a very low run-time. This can be explained by the light-weight, JavaScript and NodeJS[4]-based implementation, executed in the Node-RED run-time engine, which enables efficient processing of data flows.

The DSDs and DPDs of the non-executable Mashup Plan are transformed to executable Data Processing Nodes (eDPNs) that can be executed using a suitable execution engine. An eDPN represents an implementation, in other words a piece of code, for e.g., a Java Web Service, that executes data extraction, data processing analytics or data storage operations. The eDPNs corresponding to the DSDs are data source adapters implementing the extraction of data from a domain specific artifact model, e.g., an enterprise information system. The domain-specific artifact models are then mapped to technical data structures such as a database table, a file-based storage, or an unstructured text file. The eDPNs corresponding to the DPDs are data processing operations implemented as Java Web Services, e.g., filter or join operations or data storage operations. The resulting executable Mashup Plan is then executed by a suitable execution engine, which is scalable and cloud based. The execution engine is a dataflow, workflow or event processing engine, for example, a BPEL Workflow engine or Node-RED engine that invokes the eDPNs in the order defined in the executable Mashup Plan.

However, the run-time of data operations depend on various factors such as size of the input dataset, complexity of data, available computing resources. Therefore, the DSD/DPD-to-eDPN mapping should take these factors into consideration. This thesis proposes an approach to dynamically assign appropriate eDPNs to DSDs/DPDs, taking into consideration factors which impact the run-time efficiency of the data operations. Further, a Map-Reduce approach is proposed to perform data processing operations on big input data. In other words, data operations on small input data should be processed in-memory by the existing Web Services but data operations on big input data should be processed using Map-Reduce implementations. Map-Reduce is a parallel data processing tool used widely in areas where massive data is involved, thus, proves suitable for Data Mashup scenarios involving data intensive operations. The parallel processing of data leads to improvement in execution run-time, thus, making the Map-Reduce approach suitable for Data Mashup scenarios involving data intensive operations. Apart from the parallel processing of data, Map-Reduce has other advantages making it further suitable for Data Mashups. Some of them, as surveyed and listed in [LLC+12], are listed below:

[4] http://nodered.org/

Simple and easy to use: Map-Reduce is simple and easy to use because the programmer has to define his job with only *Map* and *Reduce* functions without having to specify physical distribution of his job across nodes.

Flexible: Map-Reduce does not have any dependency on data model and schema. Irregular and unstructured data can also be processed using Map-Reduce.

Independent of the storage: Map-Reduce is basically independent of the underlying storage layers and hence, can work with different storage layers such as BigTable [CDG+08], a distributed storage system for structured data, and others.

Fault tolerance: Map-Reduce is highly fault tolerant. For example, it is reported that Map-Reduce can continue to work in spite of an average of 1.2 failures per analysis job at Google [DG08].

High scalability: The most important advantage of using Map-Reduce is high scalability. This feature is very advantageous to achieve efficiency optimization while processing very big data sets.

4 Efficiency Optimization of data-intensive Data Mashups

This section describes the proposed approach to achieve efficient run-time execution of Data Mashups, especially in case of Data Mashups executing data-intensive operations. Operations processing huge amounts of data or data which has high complexity are referred to as *data-intensive operations* in this work. The state of the art Data Mashups fetch data from different sources, process the data using data operations such as filter and integrate the data using operations such as join, analyse. The aim of Data Mashup is to enable flexible, ad-hoc integration of heterogeneous data sources into a single output data source. This composite source of data can be very valuable for activities like data analysis in order to obtain valuable information [DM14].

However, in order to perform Data Mashups, the user oftentimes needs to posses IT skills to understand and model the data operations, i.e., the user to some extent has to understand the technical details of the data operations. This puts the domain-users/business-users at a disadvantage, since they, more often than not, are no IT experts. FlexMash with its three abstraction levels as shown in Figure 4.1, namely the modeling level, the transformation level and the execution level, is able to deal with

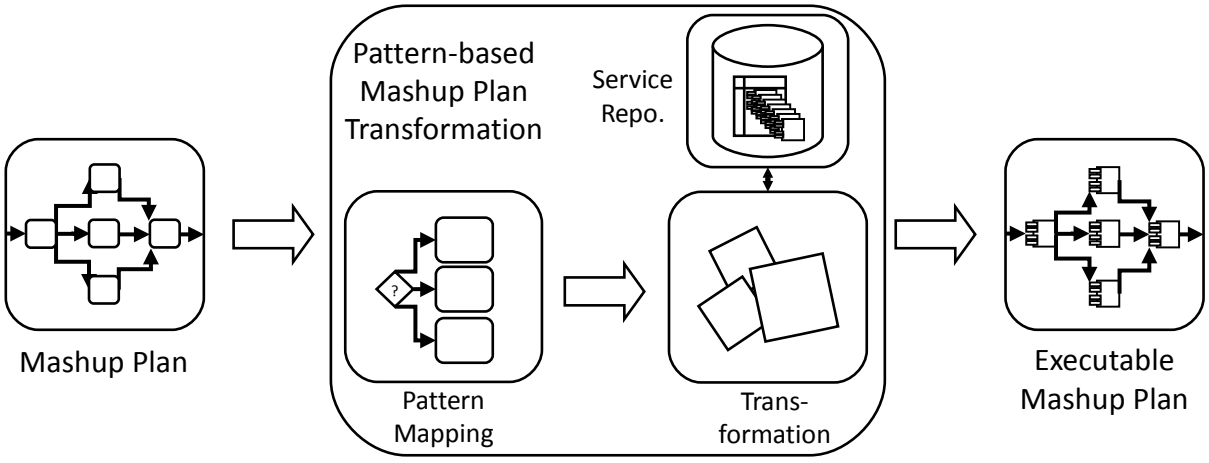

Figure 4.1: FlexMash Current Architecture [HRWM15]

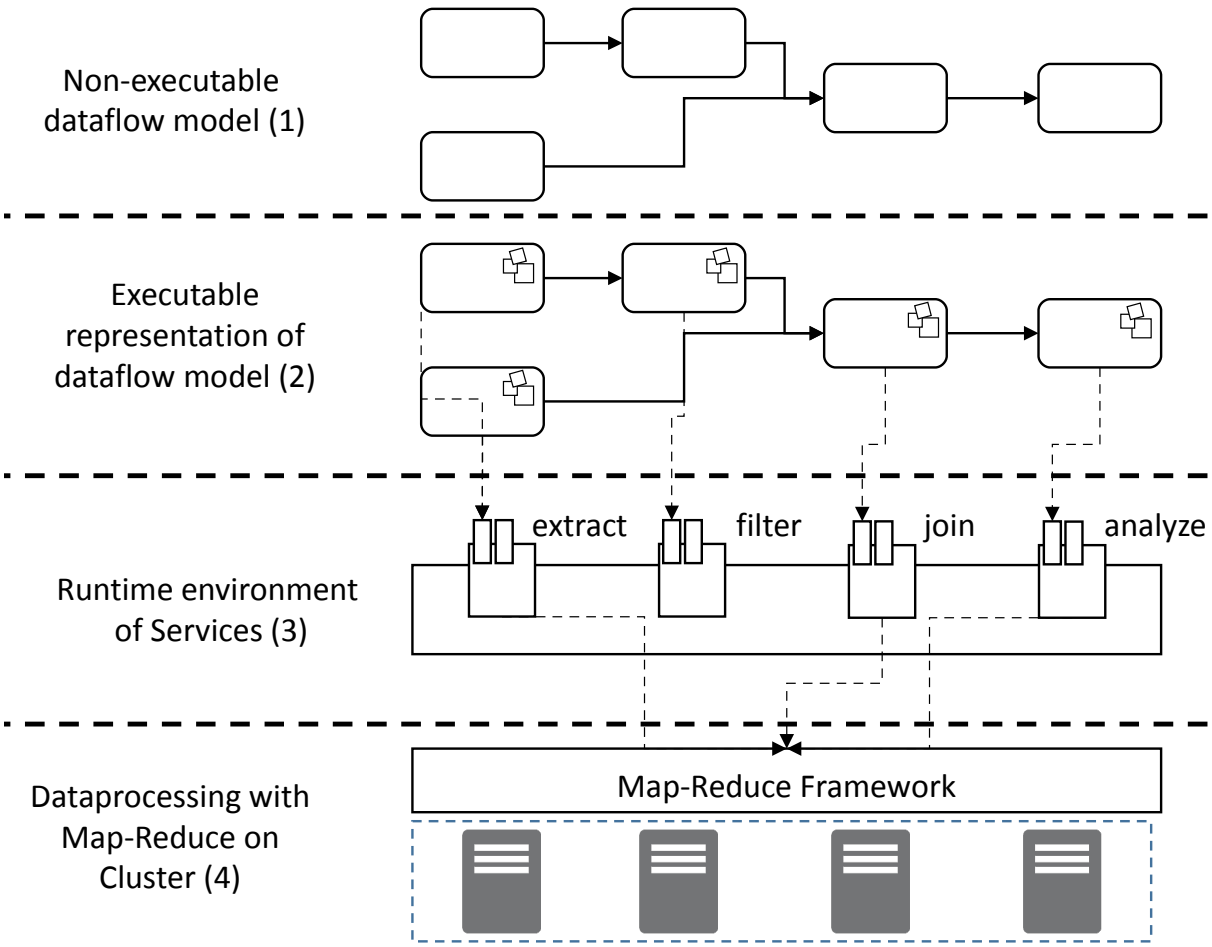

Figure 4.2: Extended FlexMash Architecture [Hir17]

this limitation. The modeling of a non-executable Mashup Plan using DSDs and the DPDs, which are domain specific representations of data sources and data operations, enable the abstraction from implementation details. The transformation patterns which facilitate the automatic execution of the Mashup Plan on a suitable execution engine enable the abstraction from the execution details [HRWM15].

However, the existing FlexMash architecture is not efficient in case of Data Mashups involving data-intensive operations, i.e., in scenarios involving large data sets or complex data. For example, a Data Mashup using a *Big Data Robust Mashup* transformation pattern will not be efficient in the current FlexMash architecture. The *Big Data Robust Mashup* transformation pattern requires that the implementation has to support the processing of huge data sets in reasonable time. But the services that currently implement the data operations are not scalable, as they process the entire input data set at one go. For very big data sets, as expected in case of *Big Data Robust Mashup*, such non-scalable implementations are inefficient. A distributed processing of the input data can enhance

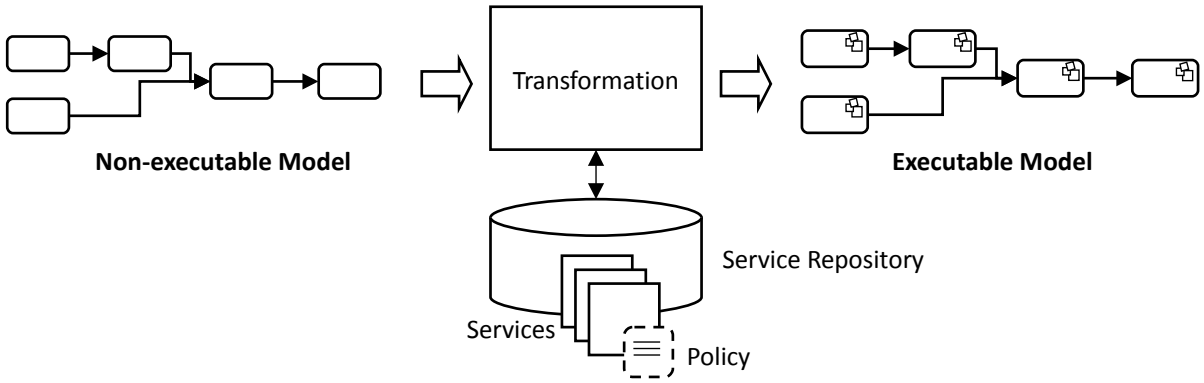

Figure 4.3: Annotating Web Services with Policies [Hir17]

the run-time efficiency of data operations involving big data. FlexMash therefore should accommodate services which process data in a parallel manner. In other words, the existing FlexMash architecture is inefficient in executing Data Mashups comprising of data-intensive data operations. To deal with this limitation, an additional step in the existing data processing architecture of the FlexMash is proposed in [Hir17]. As already mentioned above, the FlexMash data processing architecture currently consists of three levels: (i) the non-executable Data Mashup plan, (ii) an executable transformation of the modeled Mashup plan using patterns and (iii) the services executing the data operations. As shown in Figure 4.2, a fourth level, i.e, processing the data sets in a parallel manner using technologies such as Map-Reduce, is introduced in order to achieve scalability of the services. This new level enables the abstraction of the third level, i.e, the services level. Consequently, the proposed architectural change extends the existing architecture without needing to dissolve it.

Although the parallel execution of the data operations on the distributed data sets reduce the execution time of the data operations, the network communication occurring during the distribution of the data in the Map step and the collection of the results in the Reduce step leads to overhead. Hence, Map-Reduce is only then advantageous, when the reduction in data processing time is sufficiently more than the communication overhead incurred. It is therefore to be noted, that not all data operations qualify for a Map-Reduce approach. Also, it is to be noted, that the same data operation may not always qualify for Map-Reduce. A typical example could be a data operation with low run-time complexity. When such an operation is executed on a small data set, a Map-Reduce implementation will lead to an overhead and will therefore not result in any efficiency optimization. However, when the same operation is applied on a very big data set, the Map-Reduce approach will improve the run-time efficiency of the data operation because of the applied parallelism. This calls for a *dynamic decision making* depending on parameters influencing the run-time of the data operation. This decision has to be made at the time of the transformation of the non-executable Mashup plan into an executable

Listing 4.1 XML Schema to create a WS-Policy defining run-time parameters

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >

      <xs:element name="Capability" type="tCapability"/>

      <xs:complexType name="tCapability">
            <wsp:ExactlyOne>
                  <wsp:All>
                        <xs:sequence>
                              <xs:element name="DataSize" type="tValue" minOccurs="0"
                                  maxOccurs="1"/>
                              <xs:element name="DataComplexity" type="tValue"
                                  minOccurs="0" maxOccurs="1"/>
                              <xs:element name="Resources" type="tValue" minOccurs="0"
                                  maxOccurs="1"/>
                        </xs:sequence>
                  </wsp:All>
            </wsp:ExactlyOne>
      </xs:complexType>

      <xs:complexType name="tValue">
            <xs:sequence>
                  <xs:element name="value">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern value="LOW|MEDIUM|HIGH"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>

</xs:schema>
```

one, so that the executable representation consists of the appropriate services. In the context of FlexMash, the dynamic mapping of DSDs/DPDs to eDPNs should be based on the decision, which pipeline operators should be processed using Map-Reduce and which operators are to be processed by existing technologies such as Web Services. Once it is decided that an operation should be processed using Map-Reduce, corresponding Map-Reduce jobs are invoked that process the data.

This decision making process is enabled by annotating Web-Services with Policies, as shown in Figure 4.3. Web Services use policies to declare in a consistent and

Listing 4.2 Sample WS-Policy suitable for Map-Reduce

```
<wsp:Policy wsu:Id="ServiceCapability">

      <Capabilities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                  xsi:noNamespaceSchemaLocation="ws_capability.xsd">

            <wsp:ExactlyOne>
                  <wsp:All>
                        <DataSize>HIGH</DataSize>
                        <DataComplexity>LOW</DataComplexity>
                        <Resources>HIGH</Resources>
                  </wsp:All>
                  <wsp:All>
                        <DataSize>LOW</DataSize>
                        <DataComplexity>HIGH</DataComplexity>
                        <Resources>HIGH</Resources>
                  </wsp:All>
            <wsp:ExactlyOne>
      </Capabilities>
 </wsp:Policy>
```

Listing 4.3 Sample WS-Policy unsuitable for Map-Reduce

```
<wsp:Policy wsu:Id="ServiceCapability">

      <Capabilities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                  xsi:noNamespaceSchemaLocation="ws_capability.xsd">

            <wsp:ExactlyOne>
                  <wsp:All>
                        <DataSize>LOW</DataSize>
                        <DataComplexity>LOW</DataComplexity>
                        <Resources>LOW</Resources>
                  </wsp:All>
            <wsp:ExactlyOne>
      </Capabilities>
 </wsp:Policy>
```

Listing 4.4 Sample WS-Policy attachment

```
<wsdl:service name="TwitterFilterService">

      <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"
                           URI="#ServiceCapability"/>

      <wsdl:port binding="impl:TwitterFilterSoapBinding" name="TwitterFilter">
            <wsdlsoap:address
                location="http://localhost:8080/Data_Mashup/services/TwitterFilter"/>
      </wsdl:port>

</wsdl:service>
```

standardized manner what they are capable of supporting and which constraints and requirements they impose on their potential requestors. WS-Policy therefore plays a very important role in interoperability [Pap08]. A Web Service uses policies to describe its non-functional capabilities, preferences and requirements in contrast to WSDL, which describes the functional aspects of a Web Service. Thus, a separation of concerns is achieved [WCL+05]. A policy is an XML structure which consists of three components, namely policy assertions, policy expressions and policy operations. Policy assertions are the building blocks of a policy. Each assertion is an atomic representation of a requirement, capability, preference or constraint of a Web Service. A collection of such policy assertions form a policy alternative. Using policy operators, namely <All> and <ExactlyOne>, policy assertions can be logically combined in order to describe complex policy requirements. In other words, policy operators group policy assertions into policy alternatives. A policy expression is an XML representation of a Web Service Policy [Pap08]. In the context of FlexMash, the policy assertions represent the parameters which influence the run-time efficiency of the operation implemented by the service, such as the data complexity, size of the data, cost of available resources to process the data. An XML schema definition, listed in Listing 4.1, has been defined to enable correct definition of policy assertions representing the parameters that influence the run-time. Two exemplary instances of the WS-Policy are shown in Listing 4.2 and Listing 4.3 respetively. The policy in Listing 4.2 indicates that the service is suitable for Data Mashups dealing with big data or highly complex data or when there are sufficient resources at disposal for executing the operations. The policy in Listing 4.3, on the other hand, indicates that the service is suitable for Data Mashups dealing with small data or data with low complexity or when there are not enough resources at disposal for executing the operations. Thus, the WS-Policy can be used by a service requester to decide whether or not to use the service.

The illustrated Policies in Listing 4.2 and Listing 4.3 are said to be in *Normal Form*. The normal form is a straightforward representation of a policy enumerating each of

its alternatives that in turn enumerate each of their assertions. Each valid policy is contained within an <All> element and all of the alternatives are contained under a single <ExactlyOne> operator [Pap08]. The sample policy in Listing 4.2 has two policy alternatives, each of which is encapsulated by the <All> operator and contains three policy assertions. A policy can be identified and reused by using the <wsu:Id> element. An unnormalized policy is allowed and WS-Policy has normalizing algorithms to convert unnormalized policies into normal form. However, it is advised to use the normal form whenever practical [Pap08]. Therefore the normal form is used in this project.

After the policy has been defined, it needs to be associated with a *policy subject*. Policies can be attached to any of the WSDL definitions such as Message, Operation, Endpoint or Service. These are referred to as policy subjects [Pap08]. In FlexMash, the policy is associated with the Service element, also called *service policy subject*. This is illustrated in Listing 4.4. The policy thus applies to all endpoints and all operations associated with the service. The policy attachment is done using the <wsp:PolicyReference> element and is called *policy inclusion* [Pap08]. The policy inclusion strategy defines the policy as a part of the subject's definition, in this case as a part of the WSDL service definition.

The decision whether or not to use the Map-Reduce approach has to be made at the time of transformation of the modeled Mashup plan into the executable plan so that the executable plan has appropriate implementations corresponding to the modeled data operations. The SelectService implementation achieves this decision-making by comparing the parameters it obtains from the user or the service requestor (for example, the transformation pattern in the context of FlexMash) with the capabilities described by the services in their policies. This comparison is also called *policy intersection* [Pap08]. The foremost requirement for a policy comparison/intersection is that the policy needs to be in normal form, which is already satisfied in case of the policies in FlexMash. The policy alternative, in other words all the assertions of the chosen policy alternative, are compared using their qualified names. The intension is to eliminate mismatches rather than finding matches. Matching policy assertions based on their qualified names, therefore, do not necessarily mean they are equal. It simply means that these assertions can be further inspected using domain-specific means to find out if they are compatible with the requirements of the requestor. Figure 4.4 illustrates the execution of the SelectService for the *Big Data Robust Transformation Pattern*. The SelectService searches for relevant Web services stored in the Service Repository using their qualified names. The services which are found relevant are further inspected by parsing its WSDL to obtain the WS-Policy. The policy assertions are then compared to the requirements of the user/service requestor, in this case the transformation pattern. If more than fifty percent of the requirements are found to be satisfied, the inspected service is chosen as the implementation for the respective data operation and thus becomes a part of the executable Mashup plan. If multiple services qualify for the same data operation, one of them is arbitrarily chosen, in case of FlexMash the first one is selected. When no

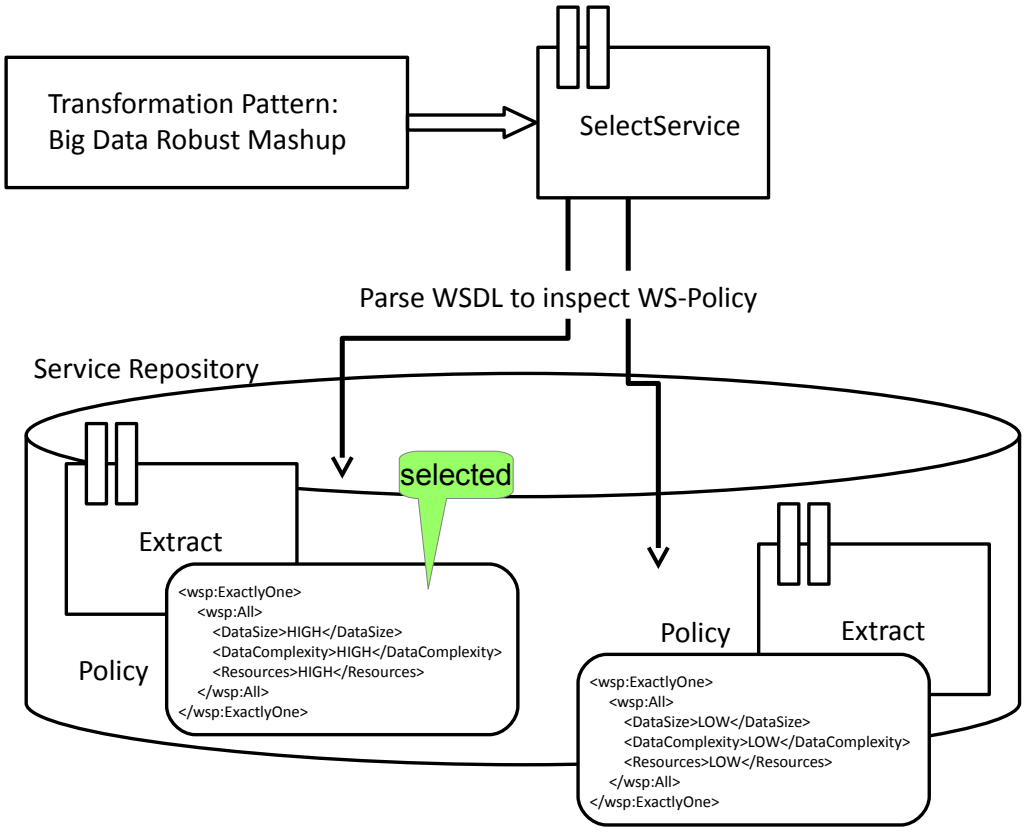

Figure 4.4: SelectService for Big Data Robust Mashup Transformation Pattern

services qualify, a default service is chosen and in this case the efficiency optimization is not guaranteed. Thus, depending on the parameters which influence the run-time of a data operation, a suitable implementation can be dynamically selected, consequently leading to efficiency optimization because not the same implementation is best suited for all kinds of Data Mashup situations.

In the following section, the run-time results of two different implementations, a Map-Reduce implementation of extract and filter operations and a regular java web implementation of extract and filter, are evaluated to show how they perform best when selected for appropriate data operations. The size of the input data is the differentiating factor and is considered for the prototypical implementation because the run-time of extract and filter operations directly depends on the size of input data. Also the Map-Reduce framework is best suited for big data and hence makes a very good implementation choice in this scenario. However, there are many different aspects which influence the run-time efficiency of data operations and suitable implementations can be created for different scenarios to improve the Mashup efficiency.

5 Prototypical Implementation and Evaluation

In the FlexMash project, the proposed approach of executing data intensive data operations using Map-Reduce is implemented for the data operations Extract and Filter. One set of Extract and Filter operations are implemented as Web Services without the Map-Reduce approach, as shown in Listing 5.1 and Listing 5.2. Currently in FlexMash, the in-memory processing of data is used for all situations irrespective of the size of the input file, complexity of the data to be processed, available resources to process the data etc. This approach is static and non-scalable leading to low performance in case of data intensive operations.

To avoid the above mentioned limitations, another set of Extract and Filter operations have been implemented using a Map-Reduce framework, as shown in Listing 5.3 and Listing 5.4. This implementation processes the data in a parallel manner using a Map and Reduce logic, thereby improving the run-time efficiency. Thus, depending on factors influencing the run-time, a decision has to be taken to either process the data using the existing implementation or using the Map-Reduce implementation. To enable this decision making, a SelectService service has been implemented, which is another java implementation, as shown in Listing 5.5. The SelectService gets the run-time requirements from the user or from any other service requester and compares them with the capabilities offered by the Web Service. These capabilities are a part of the WSDL document of the Web Service and are identified as WS-Policies. The SelectService parses the WSDL documents which are stored in a repository and compares the requirements obtained from the user/service requester with the Web Service policies mentioned in the WSDL documents, as shown in Listing 5.6, Listing 5.7 and Listing 5.8. A Web Service is selected when more than 50 percent of the capabilities match with the requirements. If multiple WSDLs satisfied the requirements, an arbitrary selection is made, i.e., the first in the search list is chosen. In a situation, when no Web Service satisfied the requirements, the default WSDL is selected and the input file is processed using the respective Web Service. This way a suitable execution is selected depending on the factors which impact the run-time capability of a data operation. Thus, the non-executable mashup plan is transformed into an executable one by dynamically choosing appropriate implementations from the repository.

Listing 5.1 In-Memory processing by Extract Operation

```
@WebService(name = "CSVExtractor")

public class CSVExtractor {

	String csvFilePath = null;
	String separator = ",";
	String line = "";

	@WebMethod(operationName = "extract")

	public String extract (@WebParam(name = "filePath") String csvFilePath) {
		JSONArray resultSet = new JSONArray();
		File file = new File(csvFilePath);

		try (BufferedReader br = new BufferedReader(new FileReader(file))) {
			String[] metaData = br.readLine().split(separator);

			while ((line = br.readLine()) != null) {
				String[] data = line.split(separator);
				JSONObject result = new JSONObject();
				for (int i = 0; i < metaData.length; i++) {
					result.put(metaData[i], data[i]);
				}
				resultSet.add(result);
			}
			return resultSet.toJSONString();
		} catch (IOException ioe) {
			ioe.printStackTrace();
		}
		return null;
	}
}
```

	File Size in MB							
Extract	**25**	**50**	**100**	**200**	**300**	**400**	**500**	**700**
without MR	4.15	8.90	17.74	40.56	79.15	107.72	129.80	191.33
with MR	17.60	18.65	19.72	25.20	27.66	35.21	41.30	54.42

Table 5.1: Run-time of extract operation with and without Map-Reduce

Listing 5.2 In-Memory processing by Filter Operation

```
@WebService(name = "Filter")
public class Filter {

	@WebMethod(operationName = "filter")

	public String filter (@WebParam(name = "extractedData") String inputJSONData,
				@WebParam(name = "criteria") String criteria) {
		JSONArray resultJSONArray = new JSONArray();
		JSONParser parser = new JSONParser();

		try {
			JSONArray jsonArray = (JSONArray) parser.parse(inputJSONData);

			for (int i = 0; i < jsonArray.size(); i++) {
				JSONObject currColumn = (JSONObject) jsonArray.get(i);
				Object[] keys = currColumn.keySet().toArray();

				for (int j = 0; j < keys.length; j++) {
					if (currColumn.get(keys[j]).equals(criteria)) {
						resultJSONArray.add(currColumn);
					}
				}
			}
			return resultJSONArray.toJSONString();
		} catch (ParseException e) {
			e.printStackTrace();
		}
		return null;
	}
}
```

	File Size in MB							
Filter	**25**	**50**	**100**	**200**	**300**	**400**	**500**	**700**
without MR	4.92	6.60	15.75	43.54	82.91	115.49	157.14	239.27
with MR	21.45	23.72	25.82	34.73	45.95	52.74	62.35	83.83

Table 5.2: Run-time of filter operation with and without Map-Reduce

Listing 5.3 Map-Reduce implementation of Extract Operation

```
public class CSVExtractorSpark implements Serializable{

      private static final long serialVersionUID = 1L;
      String separator = ",";
      String line = "";
      List<JSONObject> resultSet = new ArrayList<JSONObject>();
      JSONArray jsonResultSet = new JSONArray();
      List<String> data = new ArrayList<String>();

      public void extract (String csvFilePath, String jsonFilePath) throws IOException {

            SparkConf conf = new SparkConf().setAppName("sparkExtract");
            JavaSparkContext sc = new JavaSparkContext(conf);
            JavaRDD<String> data = sc.textFile(csvFilePath);
            final String[] header = data.first().split(separator);
            JavaRDD<JSONObject> rdd_extract_json = data.map(
                        new Function<String, JSONObject>() {
                              public JSONObject call(String line) {
                                    JSONObject result = new JSONObject();
                                    String[] fields = line.split(",");
                                    for (int i = 0; i < header.length; i++) {
                                          result.put(header[i], fields[i]);
                                    }
                                    return result;
                              }
                        } );
            rdd_extract_json.saveAsTextFile(jsonFilePath);
      }
}
```

Listing 5.4 Map-Reduce implementation of Filter Operation

```
public class FilterSpark {
      public void filter (String extractJSONOutputPath, String[] criteria) throws
          IOException {

            SparkConf conf = new SparkConf().setAppName("sparkFilter");
            JavaSparkContext sc = new JavaSparkContext(conf);
            SQLContext sql = new SQLContext(sc);
            DataFrame jsonRecordsDF = sql.jsonFile(extractJSONOutputPath);
            long count =
                jsonRecordsDF.filter(col(criteria[0]).contains((criteria[1]))).count();
            System.out.println(count);
            }
}
```

Listing 5.5 SelectService Implementation

```
public class SelectService {

	int matchedCount;
	private String dataOperation = null;
	private List<String[]> requiredCapabilities = new ArrayList<String[]>();

	// Class Constructor
	public SelectService (String dataOperation, List<String[]> requiredCapabilities) {
		this.dataOperation = dataOperation;
		this.requiredCapabilities = requiredCapabilities;
	}

	//Method to return the appropriate service
	public String selectService () {
		List<File> matchedWsdls = queryRepository(dataOperation);
		String selectedService = inspectMatchedWsdls(matchedWsdls);

		return selectedService;
	}
```

Listing 5.6 Query Service Repository

```
	/* Method to query the ServiceRepository
	 * Service Respository is queried for the dataoperation related services.
	 * We are quering a local repository here for the sake of simplicity
	 * A list of all related wsdls is returned by Service Repository
	*/
	private List<File> queryRepository (String dataOperation) {
		List<File> repositoryResponses = null;
		repositoryResponses = new ArrayList<File>();
		File[] files = new File("src/ServiceRepository").listFiles();

		for (File file : files) {
			if (file.isFile()) {
				if (file.getName().contains(dataOperation)) {
					repositoryResponses.add(file);
				}
			}
		}
		return repositoryResponses;
	}
```

Listing 5.7 Inspect matched WSDLs

```
/* Method to inspect all the wsdls returned by the ServiceRepository
 * to find the service that fulfills the required capabilities.
*/
private String inspectMatchedWsdls (List<File> matchedWsdls) {
      String matchedService = null;
      List<String> fullyMatched = new ArrayList<String>();
      List<String> partiallyMatched = new ArrayList<String>();
      for (File wsdl : matchedWsdls) {
            List<String[]> serviceCapabilities = extractServiceCapabilities(wsdl);
            //Logic to compare the requiredCapabilities and serviceCapabilities
            matchedCount = 0;
            for (String[] requirement : requiredCapabilities) {
                  for (String[] capability : serviceCapabilities) {
                        if (requirement[0].equals(capability[0])) {
                              if (requirement[1].equals(capability[1])) {
                                    matchedCount ++;
                              }
                        }
                  }
            }
            if (matchedCount == requiredCapabilities.size()) {
                  fullyMatched.add(wsdl.getName());
            } else if (matchedCount >= requiredCapabilities.size()/2) {
                  partiallyMatched.add(wsdl.getName());
            }
      }

      if (!fullyMatched.isEmpty()) {
            matchedService = fullyMatched.get(0);
      } else if (!partiallyMatched.isEmpty()) {
            matchedService = partiallyMatched.get(0);
      } else {
            matchedService = "DefaultFilter.wsdl";
      }
      return matchedService;
}
```

Listing 5.8 Extract Service Capabilities from WS-Policy

```
	/* Method to extract the service capabilities in the policy segment of wsdl.
	 * Converts the byte[] input to text file and reads the policy segment
	 * Returns a 2 dimensional array of (capability-name,capability-value) pairs
	*/

	private List<String[]> extractServiceCapabilities(File wsdl) {
		List<String[]> serviceCapabilities = new ArrayList<String[]>();
		try {
			File inputFile = new File(wsdl.getPath());
			DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
			DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
			Document doc = dBuilder.parse(inputFile);
			doc.getDocumentElement().normalize();
			Element root = doc.getDocumentElement();
			NodeList childNodes = root.getChildNodes();
			for (int temp=0; temp < childNodes.getLength(); temp++) {
			  Node node = childNodes.item(temp);
			  if (node.getNodeName() == "wsp:Policy") {
			    NodeList capaNodes = node.getChildNodes();
			      for (int idx=0; idx < capaNodes.getLength(); idx++) {
			        Node capaNode = capaNodes.item(idx);
			        if (capaNode.getNodeName() == "Capabilities") {
			            NodeList capabilities = capaNode.getChildNodes();
			            for (int id=0; id < capabilities.getLength(); id++) {
			                  Node capability = capabilities.item(id);
			                  if (capability.getNodeType() == Node.ELEMENT_NODE) {
			                     serviceCapabilities.add(new String[]
			                                   {capability.getNodeName(),
			                                    capability.getTextContent()});
			                  }
			             }
			          }
			  }
			 }
		} catch (Exception e) {
			e.printStackTrace();
		}
		return serviceCapabilities;
	}
}
```

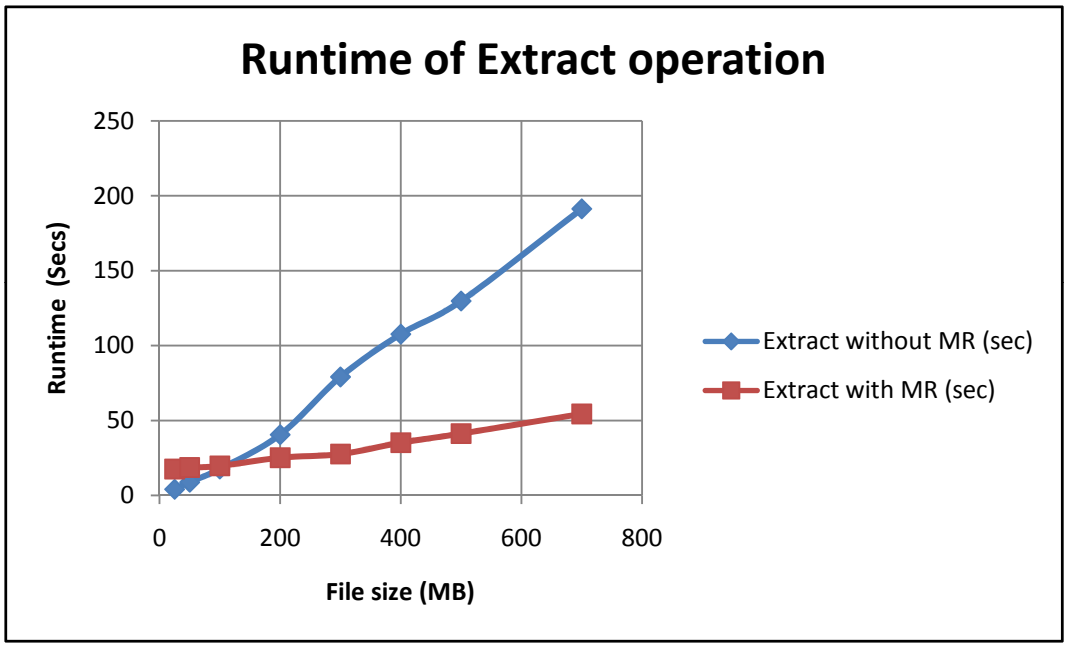

Figure 5.1: Execution run-time of Extract operation

The extract and filter operations are executed on input files of different sizes ranging from 25 MB to 2 GB. The aim is to inspect if the Map-Reduce approach leads to an efficiency improvement. Openstack provides the required infrastructure to execute the Map-Reduce jobs in the cluster. Table 5.1 lists the execution duration of the Extract operation and Table 5.2 lists the execution duration of the Filter operation on input files of different sizes. Each table lists the execution duration of both the implementations, i.e., the already existing Web Service implementation as well as the Map-Reduce implementation to enable a comparative study. For this set of results a minimum size cluster with just two nodes is used. As it is quite evident from both the tables, the in-memory implementations of extract and the filter operations are very inefficient for files larger than 200 MB. At the same time the Map-Reduce approach is very inefficient for smaller files, such as 25 MB or 50 MB. Such files must be processed using the existing Web Service implementations and the files bigger than 200 MB can be considered for the Map-Reduce implementation. In case of the Map-Reduce approach, there are two other overheads incurred apart from the communication overhead: (i) The input files need to be stored in HDFS (Hadoop Distributed File System) to be processed by Map-Reduce,

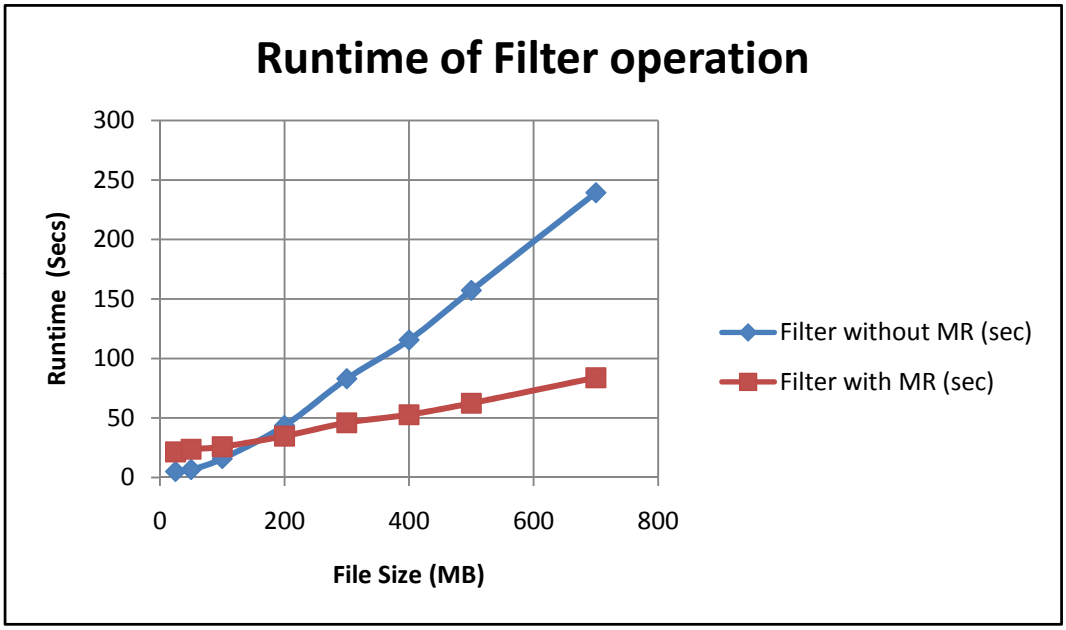

Figure 5.2: Execution run-time of Filter operation

which means that moving the files into HDFS adds to the execution overhead of the overall process. In this prototypical implementation, the moving of input files was done manually and hence the overhead is not logged. (ii) Launching the Map-Reduce jobs on the cluster needs some preparation. In this prototypical implementation, the Openstack Api (openstack4j[1]) is used to launch the Map-Reduce jobs on the cluster, thereby incurring an average overhead of 1.5 seconds. This overhead is included in the total process execution time listed in the tables above.

Figure 5.1 clearly shows that the run-time of extract operation without the Map-Reduce implementation exponentially increases as the file size increases. However the Map-Reduce implementation has more or less a linear increase in the run-time. Hence it can be concluded that for files bigger than 200 MB, the Map-Reduce approach will result in efficiency optimization. The same can also be argued for the filter operation, as shown in Figure 5.2

[1]http://openstack4j.com/

	Cluster Nodes						
Operation	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	164.90	131.90	121.40	115.40	112.10	107.00	103.25
Filter (sec)	254.00	213.60	166.30	126.10	114.20	103.89	92.35

Table 5.3: Impact of cluster size on 2 GB File

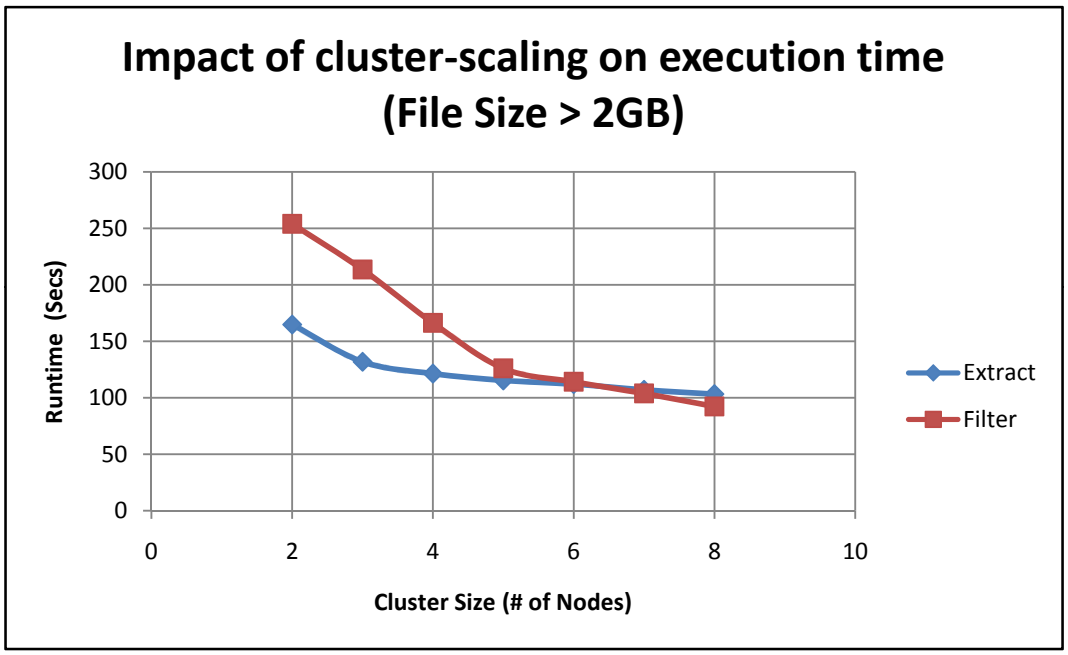

Figure 5.3: Impact of Cluster scaling on 2 GB File

	Cluster Nodes						
Operations	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	39.99	38.35	36.91	36.58	35.06	33.96	31.24
Filter (sec)	61.12	58.28	52.94	50.79	47.5	38.89	37.70

Table 5.4: Impact of cluster size on 500 MB File

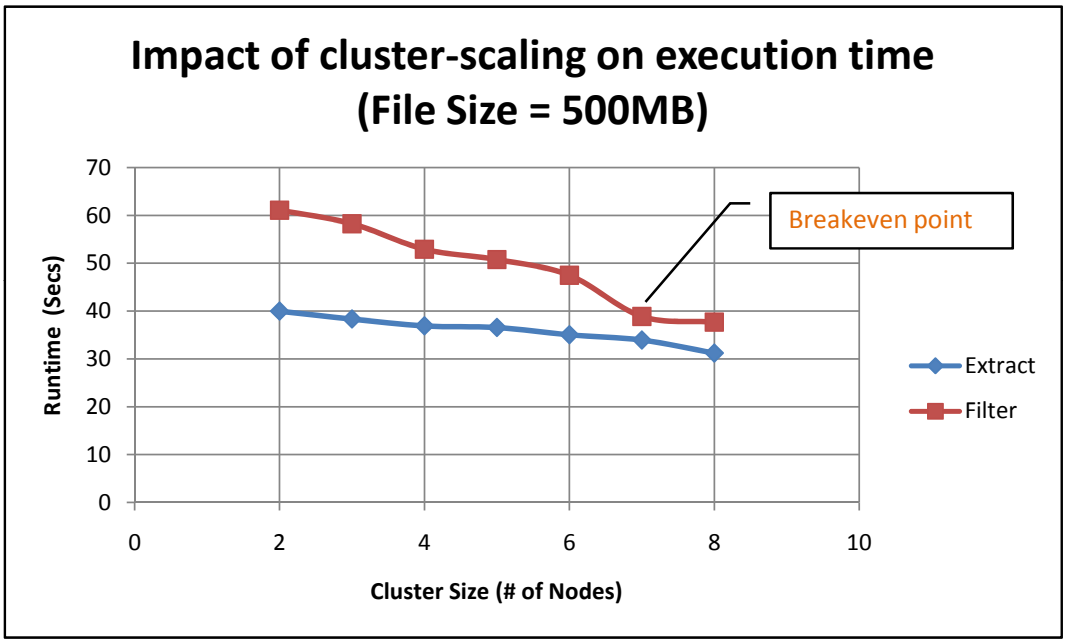

Figure 5.4: Impact of Cluster scaling on 500 MB File

	Cluster Nodes						
Operations	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	54.37	50.62	46.80	42.44	42.03	41.75	39.27
Filter (sec)	82.92	74.31	65.80	57.44	53.01	51.55	46.92

Table 5.5: Impact of cluster size on 700 MB File

	Cluster Nodes						
Operations	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	24.35	27.73	28.52	30.26	29.78	30.28	27.87
Filter (sec)	33.39	35.71	36.01	37.11	39.15	39.42	38.49

Table 5.6: Impact of cluster size on 200 MB File

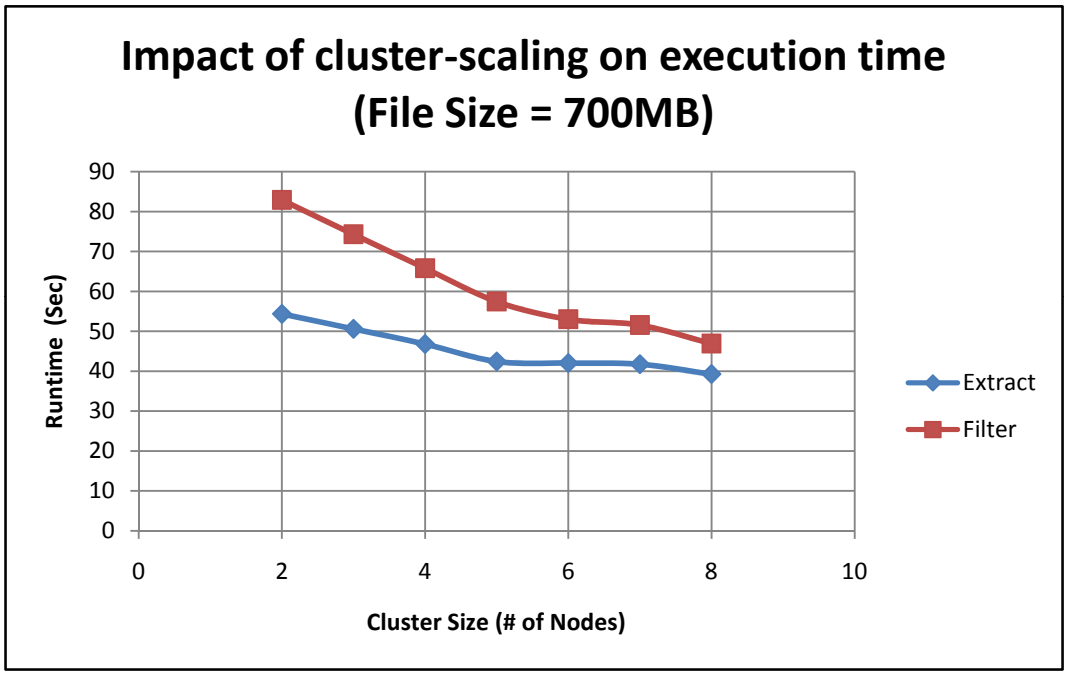

Figure 5.5: Impact of Cluster scaling on 700 MB File

	Cluster Nodes						
Operations	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	27.56	29.45	30.90	29.89	29.25	29.25	27.70
Filter (sec)	45.03	45.29	37.75	36.18	36.85	35.64	35.80

Table 5.7: Impact of cluster size on 300 MB File

	Cluster Nodes						
Operations	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Extract (sec)	34.06	34.65	33.72	33.25	34.14	33.24	33.20
Filter (sec)	51.55	51.72	50.44	42.49	41.08	37.87	34.96

Table 5.8: Impact of cluster size on 400 MB File

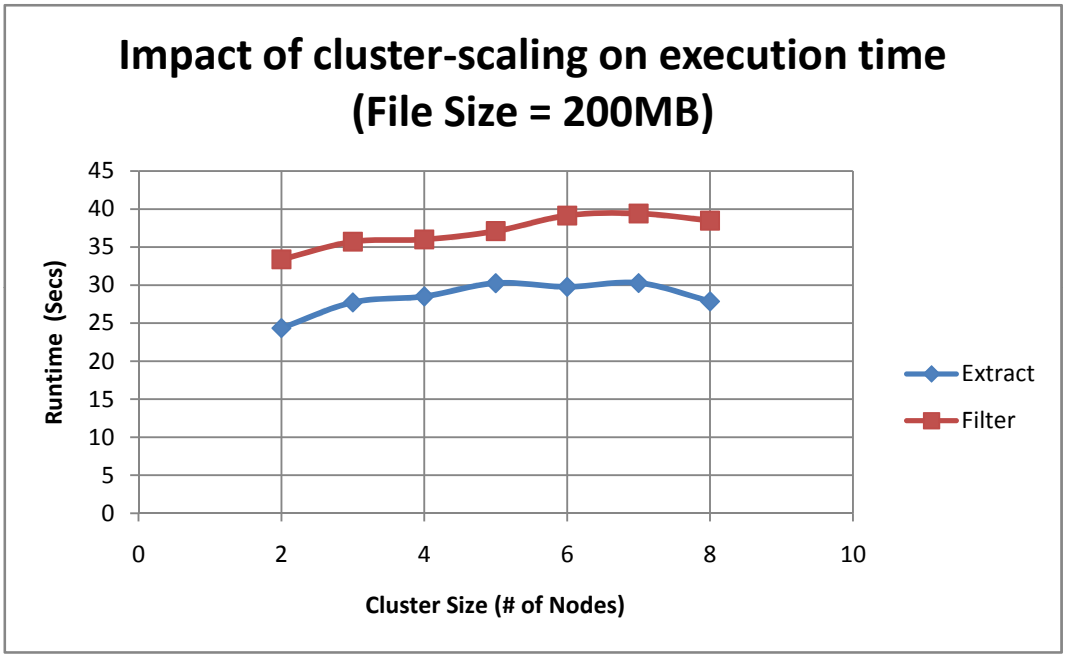

Figure 5.6: Impact of Cluster scaling on 200 MB File

The Map-Reduce implementation is further evaluated by scaling up the cluster and inspecting what impact it has on the run-time efficiency of the operations. The extract and filter operations are executed using an input file of size greater than 2 GB and the run-time durations are documented in Table 5.3. The Figure 5.3 clearly illustrates that the run-time duration of extract as well as filter operation constantly decreases as the cluster is scaled up, i.e., as the nodes in the cluster are increased, the run-time efficiency of the extract and filter operations improve. Since Map-Reduce is very suitable for processing big data sets, it can be concluded that data sets larger than 2 GB should also exhibit such efficiency improvements as in the case of the tested 2 GB file.

It is further evaluated if this trend holds for smaller data sets. As already discussed above, processing files less than 200 MB using Map-Reduce results in lower run time efficiency and hence should be processed using existing Web Services. Therefore, files greater than 200 MB were tested to find if the increase in the cluster size results in any interesting trends. As illustrated in Figure 5.4 and Figure 5.5, the 500 MB and 700 MB files show a constant improvement in execution duration when the cluster size is increased. The Table 5.4 and Table 5.5 lists the execution durations of both the

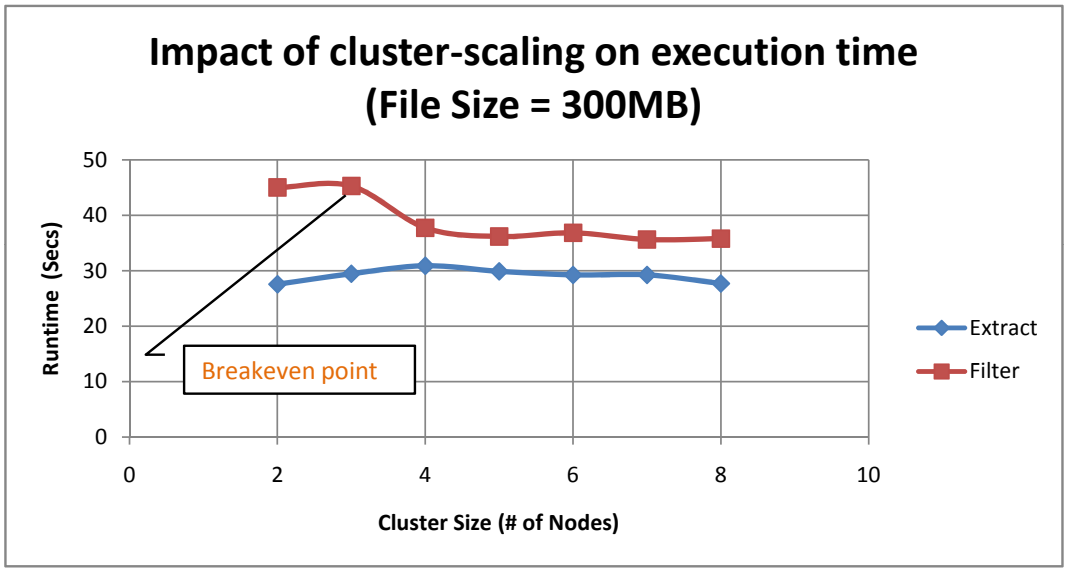

Figure 5.7: Impact of Cluster scaling on 300 MB File

operations for various cluster sizes. It can further be concluded, that for files larger than 500 MB, the execution efficiency can be further optimized by increasing the size of the cluster. A higher number of nodes result in a higher degree of parallelism, thus leading to reduction in data operation execution time. By increasing the size of a cluster the network overhead is increased. But in case of files larger than 500 MB, the increase in the communication overhead is being compensated by the reduction in execution time. However, after a certain increase in cluster size, the reduction in execution duration cannot further compensate the increase in communication overhead. Thus a negative breakeven point will be attained, after which any increase in cluster size will not lead to any further efficiency improvement. The seventh node in the cluster is a negative breakeven point in case of the filter operation in case of the 500 MB file.

On the other hand, the increase in cluster size degrades the data operation efficiency in case of low size data sets. This is because the reduction in execution time is less than the increase in communication overhead and hence unable to compensate it. As shown in Figure 5.6, the efficiency of extract and filter operations on a 200 MB input

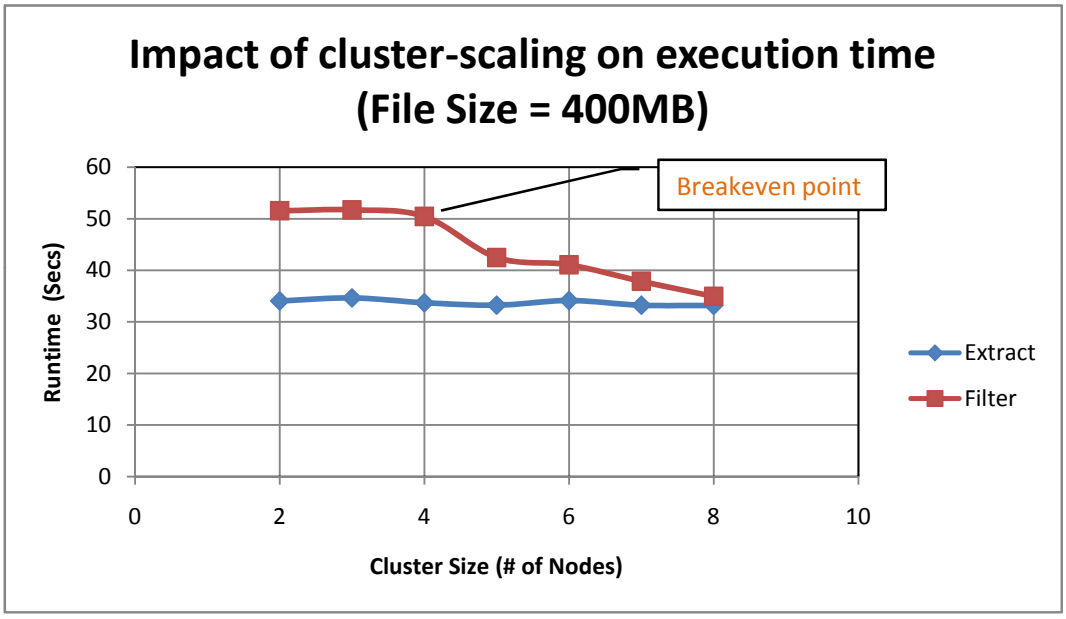

Figure 5.8: Impact of Cluster scaling on 400 MB File

file decreases when the cluster size increases. Table 5.6 lists the respective execution run time durations for both the operations.

In case of 300 MB and 400 MB files, as can be seen in Figure 5.7 and Figure 5.8, an increase in cluster size leads to neither any significant improvement nor any significant deterioration in the execution efficiency of the extract operation. The filter operation, however, shows an interesting trend. The execution efficiency initially does not show any significant change with the increase in the cluster size. However, after a breakeven point (third node in case of the 300 MB file and fourth node in case of the 400 MB file), the execution efficiency improves. This breakeven point can be referred to as a positive breakeven point after which an increase in cluster size leads to an efficiency improvement. Table 5.7 and Table 5.8 list the execution durations of both the operations for various cluster sizes.

It is therefore important to note, that an increase in the size of the cluster need not always lead to inprovement in execution efficiency. Though the parallelism in data processing is increased by adding more nodes to the cluster, but there is also an increase in the

communication overhead occuring in the Map and Reduce phases. Thus determining an appropriate size of the cluster for achieving the most efficient run-time duration is very critical and must be judiciously met.

6 Summary

Data Mashups aim at integrating disparate sources of information into a potentially more valuable composite source of information which other applications, services and processes can access and use [DM14]. The existing Data Mashup tools such as Yahoo Pipes[1], Intel MashMaker[2], IBM Infosphere Mashuphub[3], FlexMash[4], focus mainly on the usability but not on the execution efficiency. In case of FlexMash, the data processing operators, which are static and non-scalable, process the entire data set in-memory. This sort of processing is very inefficient while processing large amounts of data. A parallel processing of large data sets, for example by using Map-Reduce[5], can improve the run-time efficiency of the data operators. On the other hand, small data sets should be processed using the already existing Web Services, such as Java implementations. This selection is important because processing small data sets using Map-Reduce can lead to large communication overheads, resulting in an inefficient execution. Hence, the data operators must dynamically choose a suitable implementation depending on parameters which impact the run-time efficiency [Hir17]. The *SelectService* java implementation facilitates this dynamic decision making by inspecting the *WS-Policies* and comparing them with the *requirements* of the service requester. In the context of FlexMash, these *requirements* are the parameters which impact the execution run-time of the data operation. WS-Policy[6] is a composeable and reusable specification to deal with non-functional descriptions and quality of service aspects of Web Services [WCL+05].

In FlexMash, WS-Policies are used to define in a descriptive manner the capabilities of the Web Services, for example, if the implementation is suitable for processing large size data sets. Thus, by comparing the WS-Policies with the service requester requirements, the SelectService chooses appropriate implementations for the data operators of the Mashup. When a Map-Reduce implementation is chosen, the respective Map-Reduce jobs are triggered. Map-Reduce is a programming model and an associated implementation for processing and generating large datasets. Users specify the computation in terms

[1] https://pipes.yahoo.com/pipes/
[2] http://intel.ly/1BW2crD
[3] http://ibm.co/1Ghxv27
[4] https://github.com/hirmerpl/FlexMash
[5] http://hadoop.apache.org/
[6] https://www.w3.org/TR/ws-policy/

of a *map* and a *reduce* function, and the underlying run-time system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of network and disks [DG08].

Map-Reduce implementations of extract and filter operations on datasets smaller than 200 MB are inefficient when compared to the existing java implementations. On the other hand, for files larger than 200 MB, the execution durations of the existing java implementations increase exponentially where as the Map-Reduce implementations exhibit a linear increase. This shows that choosing an appropriate implementation for the data operators can lead to efficiency optimization. Increasing the cluster size also impacts the execution efficiency. However it does not always lead to improvement. For example, while processing the 200 MB file, increasing the cluster size beyond two nodes leads to a constant increase in run-time duration. On the other hand, a 2 GB file exhibits a constant improvement in the execution efficiency when additional nodes are added to the cluster. In case of files of intermediate size, for example 300 MB or 400 MB, interesting trends can be observed. The execution efficiency deteriorates with the increase in cluster size until a positive breakeven point is attained at the third and fourth node respectively, after which the execution efficiency improves. On the other hand, in case of the 500 MB file, especially in case of the filter operation, the execution efficiency improves with the increase in the cluster size until a negative breakeven point is attained at the seventh node, after which it does not improve any further. Thus, varying the cluster size in order to improve the execution efficiency is a critical decision and must be taken judiciously. Further research work is needed to establish a mathematical model which can enable this decision making.

7 Future Works

It is evident from the current test results that parallel processing of data sets leads to a definite optimization of run-time efficiency in case of files larger than 200 MB. Hence in this thesis, it is concluded, that files larger than 200 MB should be considered for processing using technologies such as Map-Reduce. Further testing can be carried out using files ranging between 100 and 200 MB to more precisely determine the lower limit, beyond which the Map-Reduce approach leads to efficient run-time. It is also interesting to find if there exists an upper cutoff limit. In other words, more rigorous testing can be carried out to find out if the parallel processing does not lead to any efficiency optimization beyond a certain file size.

An increase in cluster size exhibits interesting and different trends for different file sizes. Further rigorous testing with multiple files of varying sizes can help in discovering interesting patterns, which can facilitate deeper analysis and precise conclusions regarding the impact of cluster size on the data processing efficiency. Breakeven points must also be further analyzed using more test results to find patterns which can provide insights regarding the impact of cluster size on execution efficiency. These trends also vary for different data operations. More data operations should be implemented using Map-Reduce and consequently tested to analyze the reasons behind this difference. Further analysis must be done to establish a mathematical model for determining the most appropriate cluster size for a particular data operation.

Bibliography

[12] *Fachstudie MapReduce: eine vergleichende Analyse aktueller Implementierungen*. 2012, Online–Ressource. URL: http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-78113 (cit. on p. 26).

[AIS78] C. Alexander, S. Ishikawa, M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. Oxford University Press, 1978. URL: https://books.google.de/books?id=lIF3mwEACAAJ (cit. on p. 20).

[CDG+08] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber. "Bigtable: A Distributed Storage System for Structured Data." In: *ACM Trans. Comput. Syst.* 26.2 (June 2008), 4:1–4:26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816. URL: http://doi.acm.org/10.1145/1365815.1365816 (cit. on p. 32).

[DG08] J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492 (cit. on pp. 32, 58).

[DM14] F. Daniel, M. Matera. *Mashups - Concepts, Models and Architectures. Data-Centric Systems and Applications*. Springer Heidelberg, 2014. ISBN: 978-3-642-55048-5. DOI: 10.1007/978-3-642-55049-2 (cit. on pp. 20–22, 33, 57).

[FBB+14] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann. "From Pattern Languages to Solution Implementations." In: *Proceedings of the 6th International Conference on Pervasive Patterns and Applications*. Xpert Publishing Services (XPS), 2014, pp. 12–21 (cit. on p. 21).

[HB17] P. Hirmer, M. Behringer. "FlexMash 2.0 – Flexible Modeling and Execution of Data Mashups." Englisch. In: vol. 696. Communications in Computer and Information Science. Springer International Publishing, Jan. 2017, pp. 10–29. ISBN: 978-3-319-53174-8. DOI: 10.1007/978-3-319-53174-8. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INBOOK-2017-01&engl=0 (cit. on pp. 23, 24).

[Hir17] P. Hirmer. “Effizienz-Optimierung daten-intensiver Data Mashups am Beispiel von Map-Reduce.” Deutsch. In: *Proceedings der Datenbanksysteme für Business, Technologie und Web (BTW), 17. Fachtagung des GI-Fachbereichs, Workshopband*. Ed. by B. Mitschang, N. Ritter, H. Schwarz, M. Klettke, A. Thor, O. Kopp, M. Wieland. Vol. P-266. LNI. Stuttgart: Gesellschaft für Informatik (GI), Mar. 2017, pp. 111–116. ISBN: 978-3-88579-660-2. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2017-11&engl=0 (cit. on pp. 34, 35, 57).

[HRWM15] P. Hirmer, P. Reimann, M. Wieland, B. Mitschang. “Extended Techniques for Flexible Modeling and Execution of Data Mashups.” Englisch. In: *Proceedings of the 4th International Conference on Data Management Technologies and Applications (DATA)*. Ed. by M. Helfert, A. Holzinger, O. Belo, C. Francalanci. Colmar: SciTePress, July 2015, pp. 111–122. ISBN: 978-989-758-103-8. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-33&engl=0 (cit. on pp. 15, 29, 30, 33, 34).

[LLC+12] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, B. Moon. “Parallel Data Processing with MapReduce: A Survey.” In: *SIGMOD Rec.* 40.4 (Jan. 2012), pp. 11–20. ISSN: 0163-5808. DOI: 10.1145/2094114.2094118. URL: http://doi.acm.org/10.1145/2094114.2094118 (cit. on pp. 25–28, 31).

[Pap08] M. P. Papazoglou. *Web Services Principles and Technology*. Pearson Education Limited, 2008. ISBN: 978-0-321-15555-9 (cit. on pp. 38, 39).

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. ISBN: 0131488740. DOI: 10.1.1/jpb001 (cit. on pp. 25, 38, 57).

All links were last followed on May 21, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature