Reducing Data Transfer in Service-Oriented Architectures: The Circulate Approach

Adam Barker, Jon B. Weissman, *Senior Member*, *IEEE*, and Jano I. van Hemert

Abstract—As the number of services and the size of data involved in workflows increases, centralized orchestration techniques are reaching the limits of scalability. When relying on web services without third-party data transfer, a standard orchestration model needs to pass all data through a centralized engine, which results in unnecessary data transfer and the engine to become a bottleneck to the execution of a workflow. As a solution, this paper presents and evaluates Circulate, an alternative service-oriented architecture which facilitates an orchestration model of central control in combination with a choreography model of optimized distributed data transport. Extensive performance analysis through the PlanetLab framework is conducted on a web service-based implementation over a range of Internet-scale configurations which mirror scientific workflow environments. Performance analysis concludes that our architecture's optimized model of data transport speeds up the execution time of workflows, consistently outperforms standard orchestration and scales with data and node size. Furthermore, Circulate is a less-intrusive solution as individual services do not have to be reconfigured in order to take part in a workflow.

Index Terms—Service-oriented architecture, orchestration, choreography, workflow optimization.

1 Introduction

Many problems at the forefront of science, engineering and medicine require the integration of large-scale data and computing. The majority of these data sets are physically distributed from one another, owned and maintained by different institutions, scattered throughout the globe [15]. Scientists and engineers require the ability to access, compose, and process these distributed data sets in order to discover correlations, enable decision making and ultimately progress scientific discovery.

In order to integrate software and data, academia and industry have gravitated toward service-oriented architectures. Service-oriented architectures are an architectural paradigm for building software applications from a number of loosely coupled distributed services. This paradigm has seen wide spread adoption through the web services approach, which has a suite of simple standards (e.g., XML, WSDL, and SOAP) to facilitate interoperability.

These core standards do not provide the rich behavioral detail necessary to describe the role an individual service plays as part of a larger, more complex collaboration. Coordination of services is often achieved through the use of workflow technologies. As defined by the Workflow Management Coalition [16], a workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant (a resource either human or machine) to another for action, according to a set of procedural rules. Workflow is usually specified from the view of a single participant using centralized *orchestration* or from a global perspective using decentralized *choreography*.

Orchestration languages explicitly describe the interactions between services by identifying messages, branching logic, and invocation sequences. Orchestrations are described from the view of a single participant, which can be another service. Therefore, a central process always acts as a controller to the involved services. The vast majority of workflow tools are based on orchestrating services through a centralized workflow engine: the Business Process Execution Language (BPEL) [29] is the current de facto standard orchestration language. Workflow tools based on a centralized enactment engine can easily become a performance bottleneck for service-oriented workflows: all data are routed via the workflow engine, these data consume network bandwidth and overwhelm the central engine which becomes a bottleneck to the execution of a workflow. Instead, a solution is desired that permits data output from one service to be forwarded directly to where it is needed at the next service in a workflow.

Choreography on the other hand is more collaborative in nature. A service choreography is a description of the externally observable peer-to-peer interactions that exist between services, therefore choreography does not typically rely on a central co-ordinator. Refer to [3] for a summary of the differences. By adopting a choreography model, the output of a service invocation can be passed directly to where it is required, as input to the next service in the workflow; not through a centralized workflow engine as is the case with orchestration. However, although optimal in terms of data transfer, in practice, the design process and execution infrastructure for service choreography models are inherently much more complex than orchestration.

- *A. Barker is with the School of Computer Science, University of St. Andrews, Jack Cole Building, North Haugh, St. Andrews, Fife KY16 9SX, United Kingdom. E-mail: adam.barker@st-andrews.ac.uk.*
- *J.B. Weissman is with the Department of Computer Science and Engineering, University of Minnesota, Twin Cities, 4-192 Keller Hall, 200 Union St. S.E., Minneapolis, MN 55455. E-mail: jon@cs.umn.edu.*
- *J.I. van Hemert is with Optos, Queensferry House, Carnegie Campus, Enterprise Way, Dunfermline, Scotland KY11 8GR, United Kingdom. E-mail: jvanhemert@optos.com.*

Manuscript received 8 June 2010; revised 12 Jan. 2011; accepted 22 Mar. 2011; published online 1 Apr. 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2010-06-0085.
Digital Object Identifier no. 10.1109/TSC.2011.23.

Decentralized control brings a new set of problems, which are the result of message passing between asynchronous distributed and concurrent processes. Furthermore, current choreography techniques are invasive, in that each individual service needs to be reengineered in order to take part in a choreography. There are relatively few decentralized choreography languages and even fewer implementations (discussed further in Section 8); the most prevalent being the Web Services Choreography Description Language (WS-CDL) [20].

This paper presents and evaluates the Circulate architecture, which sits in between pure orchestration (completely centralized) and pure choreography (completely decentralized). This centralized control flow, distributed data flow model maintains the robustness and simplicity of centralized orchestration but facilities choreography by allowing services to transfer data among themselves, without the complications associated with modeling and deploying service choreographies. The Circulate architecture reduces data transfer between services (which don't contain functionality for third-party data transfer), which in turn speeds up the execution time of workflows and removes the bottlenecks associated with centralized orchestration.

1.1 Paper Contributions

This paper makes the following core contributions.

1.1.1 Hybrid Architecture

Circulate is a hybrid between orchestration and choreography techniques: This model maintains the robustness and simplicity of centralized orchestration but facilities choreography by allowing web services to transfer data among themselves. Importantly, the Circulate architecture is a general architecture and can therefore be implemented using different technologies and integrated into existing systems. However, in this paper we will focus on an implementation based on web services, which is used as the basis for our Internet-scale evaluation.

1.1.2 Internet-Scale Evaluation

By avoiding the need to pass large quantities of intermediate data through a centralized server, we demonstrate through Internet-scale experimentation how the Circulate architecture reduces data transfer and therefore speeds up the execution time of a workflow. Our evaluation demonstrates how Circulate scales in terms of data size and node size across a range of common workflow topologies.

1.1.3 Less-Intrusive Solution

In contrast with current service choreography techniques, Circulate is a less-intrusive solution. Our architecture is decoupled from the services they interact with and can be deployed without disrupting existing infrastructure; this means that services do not have to be altered before execution.

The remainder of this paper is structured as follows: Section 2 introduces Circulate by discussing the architecture, a web services implementation and providing a concrete example. Section 3 extracts a set of recurring workflow patterns which will be referred to throughout the remainder of this paper. Section 4 discusses the experimental set up used as the basis for the performance analysis across Local Area Network (LAN) and Internet-scale network configurations; a cross product of workflow pattern, node size, and network configuration.

Section 5 presents the results from our LAN configuration, covering the remote LAN case, where all services are deployed within a LAN but the workflow engine is remote (common cloud configuration) and the local LAN case where both the services and engine are deployed on the same LAN. Section 6 presents the results from our Internet-scale configurations, executed over the PlanetLab network, these experiments are broken down into national (all nodes in the same country), continental (all nodes in the same continent), and world wide configurations. In Section 7, the Circulate architecture is applied to an end-to-end application, Montage [17], a benchmark in the High Performance Computing community. Section 8 presents related work covering: decentralized choreography languages, data flow optimization techniques, and Grid frameworks which contain third-party data transfers. Last, Section 9 presents conclusions and future work.

2 The Circulate Architecture

This section describes the Circulate architecture's hybrid model, web services implementation, API, and a corresponding example of use.

2.1 Circulate Actors

The Circulate architecture is a hybrid between orchestration and choreography techniques. In order to provide web services with the required functionality proxies are introduced. A proxy is a lightweight middleware that provides a gateway and standard API to web service invocation. Proxies are less-intrusive than existing choreography techniques as individual services do not have to be reconfigured in order to take part in a workflow.

Proxies are controlled through a centralized workflow engine, running an arbitrary workflow language; allowing standards-based approaches, and tooling to be utilized. Proxies exchange references to data with the workflow engine and pass actual data directly to where they are required for the next service invocation. This allows the workflow engine to monitor the progress and make changes to the execution of a workflow.

To utilize a proxy, a web service must first be registered. The Circulate actors and interactions are illustrated by Fig. 1 and described below.

2.1.1 Engine to Proxy ($\mathrm{E} \rightarrow \mathrm{P}$) Interaction

In the standard orchestration model, the workflow engine interacts directly with all web services, which for the remainder of the paper we will denote the engine to web service interaction ($\mathrm{E} \rightarrow \mathrm{S}$). Using the Circulate architecture, the workflow engine remains the centralized orchestrator for the workflow, however the task of service invocation is delegated to a proxy ($\mathrm{E} \rightarrow \mathrm{P}$).

2.1.2 Proxy-Web Service ($\mathrm{P} \rightarrow \mathrm{S}$) Interaction

Proxies neither create web service requests, nor do they utilize their responses. Proxies invoke services on behalf of

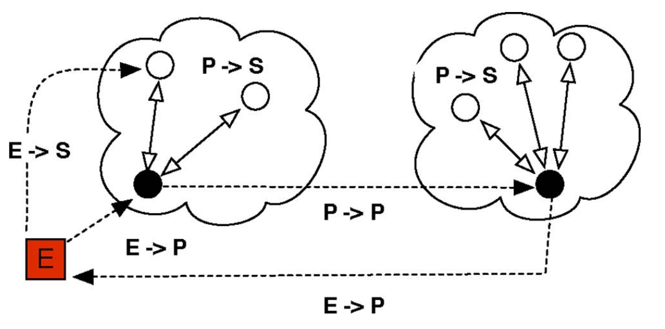


Fig. 1. Actors and interactions of the circulate architecture. Web services are represented by hollow circles, proxies by solid circles, the workflow engine as a square, dashed lines are WAN hops, and solid lines LAN hops.

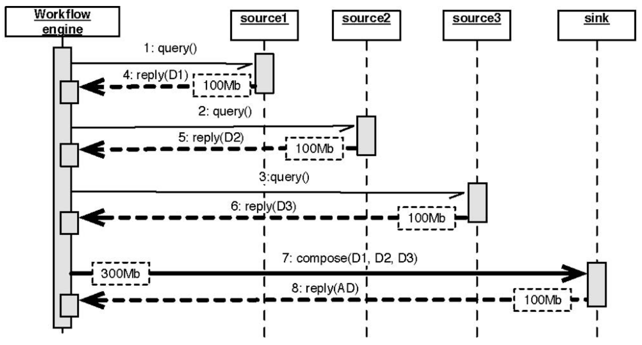


Fig. 2. UML Sequence diagram-orchestrated fan-in.

a workflow engine, store the result (if required), and return a reference to the workflow engine.

2.1.3 Proxy to Proxy ($P \rightarrow P$) Interaction

Proxies invoke web services on behalf of a workflow engine, instead of sending the results of a service invocation back to the workflow engine, they can be stored (if required) within the proxy. In order for a workflow to progress, i.e., the output of a service invocation is needed as input to another service invocation, proxies pass data among themselves, moving it closer to the source of the next web service invocation.

Proxies are ideally installed as "near" as possible to enrolled web services; by near we mean in terms of network distance, so that the communication overhead between a proxy and a web service is minimized. Depending on the preference of an administrator, a proxy can be responsible for one web service, 1:1 or multiple web services, 1:N. Although it is ideal to place a proxy as closely as possible to an enrolled service (e.g., within the same network domain) it may not always be possible due to the network policy of a particular organization. Performance benefit can still be accrued simply by harnessing the connectivity of proxies scattered across a network; this is demonstrated throughout our performance analysis across real networks, discussed further in Sections 5, 6, and 7.

2.2 Web Services Implementation

WS-Circulate is implemented using a combination of Java and the Apache Axis web services toolkit [2]. Proxies are simple to install and can be configured remotely, no specialized programming needs to take place in order to exploit their functionality. The only changes required are

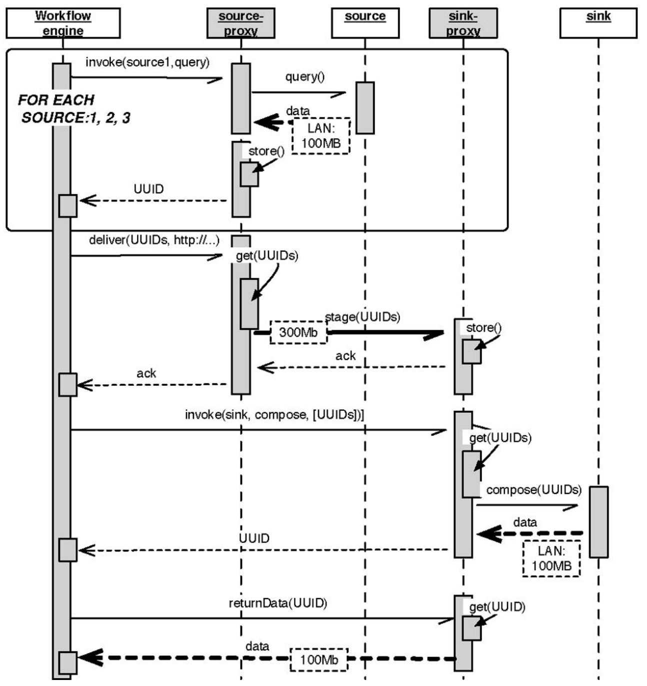


Fig. 3. UML Sequence diagram-Circulate fan-in.

to the workflow specification itself which invokes methods on the proxy rather than the services directly. WS-Circulate is multithreaded and allows several applications to invoke methods concurrently. Results from web service invocations are stored at a proxy by tagging them with a Universally Unique Identifier (UUID) and writing them to disk. There is an assumption that there is sufficient disk space and that storage is temporary, which the proxy can clean up/delete afterwards. Proxies are made available through a standard WSDL interface. This interface contains the following operations: `invoke`, `stage`, `returnData`, `flushTempData`, `addService`, `removeService`, `listOps`, `list-OpParams`, `listOpReturnType`, and `listServices`. Full details of the API and implementation of the proxy can be found in a complementary paper [4]. In order to simplify the development and deployment, issues of security have not been taken into account, this is left to future work.

2.3 Circulate Example

In order to demonstrate the Circulate architecture and API, consider the following simple scenario (an example of a fan-in pattern, discussed further in Section 3): three `sources` are queried for data via web service interfaces, these data are combined and used as input to a final `sink` service, which processes these data and returns a results set. Using UML Sequence diagram notation, standard orchestration is illustrated by Fig. 2 and Circulate by Fig. 3. Thicker arrows represent data movement, data sizes are arbitrary and used for illustrative purposes only.

With reference to Fig. 3, the first step in the workflow pattern involves making an invocation to the three source web services `source1-source3`. Instead of contacting the service directly, a call is made to a proxy (`source-proxy`) which has been installed on the same server as the `source1` web service. The proxy invokes the `query` operation on the `source1` service, the output is passed back to the proxy, tagged with a UUID (for reference later, e.g., retrieval, deletion, etc.) and stored. The UUID (not actual data) is returned to the workflow engine. This process is repeated (either serially or in parallel) for `source2` and `source3` which could be served through the same proxy or an independent proxy.

The output from the web service invocations are needed as input to the next service in the workflow, in this case the `sink` web service. The workflow engine invokes the `deliver` operation on the `source-proxy` passing in the three UUID references along with the WSDL address of the `sink-proxy`. The `source-proxy` retrieves the stored data and transfers it across the network by invoking a `stage` operation on `sink-proxy`. Data are then stored at `sink-proxy`, if successful an acknowledgement message is sent back to `source-proxy` which is returned to the workflow engine.

The final stage in the workflow pattern requires using the output from the first three services as input to the `sink` web service. The workflow engine passes the name of the service (`sink`) and operation (`compose`) to invoke and the UUID references, which are required as input. The proxy then moves the data across the network and invokes the `compose` operation on the `sink` service. The output is again stored locally on the proxy and a UUID reference generated and passed back to the workflow engine. The workflow engine can then retrieve actual data from the proxy when necessary using the `returnData` operation.

3 Workflow Patterns

As with software design patterns, workflow patterns refer to recurrent problems and proven solutions in the development of workflow applications. There is a large body of workflow patterns research detailing a comprehensive set of patterns from both a control flow and data flow perspective, the most prevalent being the work by van der Aalst and Hofstede et al. [1] and the Service Interaction Patterns set by Barros et al. [6], a collection of 13 recurring patterns derived from insights into business-to-business transaction processing.

Workflows in the scientific community are commonly modeled as Directed Acyclic Graphs (DAGs), formed from a collection of vertices (units of computation) and directed edges. DAGs present a *dataflow view*, here data are the primarily concern, workflows are constructed from data processing (vertices) and data transport (edges). DAGs may be used to model processes in which information flows in a consistent direction through a network of processors. The Genome Analysis and Database Update system (GADU) [26], the Southern California Earthquake Centre (SCEC) [13] CyberShake project, and the Laser Interferometer Gravitational-Wave Observatory (LIGO) [28] are all examples of High Performance Computing applications composed using DAGs.

OMII-Taverna [25] is an example of a popular tool used in the life sciences community in which workflows are represented as DAGs and executed using the service-oriented paradigm.

This paper focuses purely on optimizing service-oriented workflows, where services are relatively simple and are not equipped to handle third-party transfers. For the remainder of this paper we take inspiration (i.e., patterns, input-output data relationships, scenarios) from DAG-based workflows but stress that we are focused purely on optimizing service-oriented workflows.

3.1 Patterns

From a DAG one can extract a number of isolated workflow patterns (sequence, fan-in, and fan-out) which will be used as the basis for performance analysis throughout the remainder of this paper. It is important to note these patterns can be considered primitive or isolated patterns, many isolated patterns can in combination, form a macro-pattern, e.g., a fan-in followed by a fan-out. This situation will be addressed when we discuss the Montage application in Section 7.

DAG-based workflows are not the only possible representation for workflows, however, they are used as the basis for evaluation in this paper as they are commonly used to represent scientific workflows.

Before we discuss each pattern we introduce the following mathematical notation which is used to provide a concrete representation of each pattern:

- P_j : proxy.
- S_i : service.
- $\mathrm{P}(S_i)$: proxy for service S_i.
- $\underset{i,j}{\overset{\leftrightarrow}{C}}$: round-trip communication cost between entities i and j.
- $\vec{C}_{i,j}$: one-way communication cost between entities i and j.
- n represents the number of services. S_{FI} is the fan-in service, where all sources are sent, S_{FO} is the fan-out service, where source data are extracted.
- E : workflow engine.
- $TC_{pat,nc\|circ}$: total communication cost of executing a workflow pattern (seq, fo, fi—sequence, fan-out, or fan-in, respectively) in Circulate (circ) or noncirculate (nc), i.e., standard orchestration architectures.

A mathematical representation of each pattern is provided in Fig. 4 and illustrated further by Fig. 5. We have chosen this method in order to identify generic trends of recurring patterns, which are both commonplace and reflect a realistic representation of how workflows are represented and executed. Complementing the pattern-based evaluation we also execute an end-to-end application, discussed in Section 7. With reference to Figs. 4 and 5 each pattern will now be explained in detail:

3.1.1 *Sequential Pattern*

This pattern involves the chaining of services together, where the output of one service invocation is used directly as input to another, i.e., serially. Data are sent from the workflow engine to the first service in the chain and returned from the final service in the chain to the workflow engine.

For the standard orchestration model ((1), phase 1 of Fig. 5) $TC_{seq,nc}$ involves round-trip communications ($\overset{\leftrightarrow}{C}$) between services $S_1,\ldots,Sn$. For the Circulate architecture

$$TC_{seq,nc} = \sum_{S_i=1}^{n} \left(\overleftrightarrow{C}_{E,S_i}\right) \tag{1}$$

$$TC_{seq,circ} = \vec{C}_{E,P(S_1)} + \sum_{S_i=1}^{n} \left(\overleftrightarrow{C}_{P(S_i),S_i}\right) + \sum_{S_i=1}^{n-1} \left(\vec{C}_{P(S_i),P(S_{i+1})}\right) + \vec{C}_{P(S_n),E} \tag{2}$$

$$TC_{fi,nc} = \sum_{S_i=1}^{n} \left(\vec{C}_{S_i,E}\right) + \overleftrightarrow{C}_{E,S_{FI}} \tag{3}$$

$$TC_{fi,circ} = \sum_{S_i=1}^{n} \left(\overleftrightarrow{C}_{P(S_i),S_i}\right) + \sum_{S_i=1}^{n} \left(\vec{C}_{P(S_i),P(S_{FI})}\right) + \overleftrightarrow{C}_{P(S_{FI}),S_{FI}} + \vec{C}_{P(S_{FI}),E} \tag{4}$$

$$TC_{fo,nc} = \vec{C}_{S_{FO},E} + \sum_{S_i=1}^{n} \left(\overleftrightarrow{C}_{E,S_i}\right) \tag{5}$$

$$TC_{fo,circ} = \overleftrightarrow{C}_{P(S_{FO}),S_{FO}} + \sum_{S_i=1}^{n} \left(\vec{C}_{P(S_{FO}),P(S_i)}\right) + \sum_{S_i=1}^{n} \left(\overleftrightarrow{C}_{P(S_i),Si}\right) + \sum_{S_i=1}^{n} \left(\vec{C}_{P(S_i),E}\right) \tag{6}$$

Fig. 4. Equations modeling the standard orchestration and Circulate case for sequence (1 and 2), fan-in (3 and 4), and fan-out (5 and 6) patterns. Control flow is omitted.

((2), phase 2 of Fig. 5) $TC_{seq,circ}$ is calculated as follows: initial data are sent one-way ($\vec{C}$) from the workflow engine E to the first proxy $P(S_1)$, then round-trip ($\overleftrightarrow{C}$) between all proxies and services $P(S_1), S_1 \ldots$ $P(S_n), S_n$, one-way ($\vec{C}$) between $n-1$ proxies, and finally one-way ($\vec{C}$) between the final proxy $P(S_n)$ and the workflow engine E.

3.1.2 Fan-in Pattern

The fan-in pattern explores what happens when data are gathered from multiple distributed sources, concatenated, and sent to a service acting as a sink. Multiple services are invoked with a control flow (no data are sent) message asynchronously, in parallel, data are returned from each service. Once data have been received from all source services it is concatenated and sent to the sink service as input, which in turn returns final output data to the workflow engine.

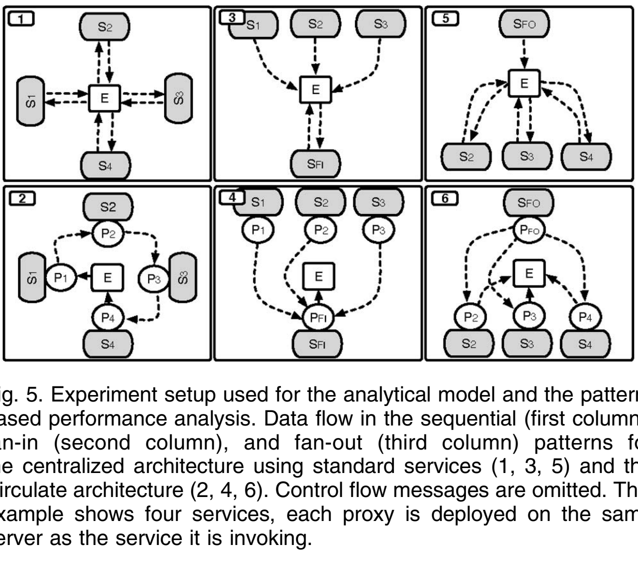


Fig. 5. Experiment setup used for the analytical model and the pattern-based performance analysis. Data flow in the sequential (first column), fan-in (second column), and fan-out (third column) patterns for the centralized architecture using standard services (1, 3, 5) and the Circulate architecture (2, 4, 6). Control flow messages are omitted. This example shows four services, each proxy is deployed on the same server as the service it is invoking.

For the standard orchestration model ((3), phase 3 of Fig. 5) $TC_{fi,nc}$ involves one-way ($\vec{C}$) communications between all sources of data $S_1 \ldots S_n$ and the workflow engine E, all source data are then sent round-trip ($\overleftrightarrow{C}$) between the workflow engine and the final sink service S_{FI}.

For the circulate architecture ((4), phase 4 of Fig. 5) $TC_{fi,circ}$ involves round trip ($\overleftrightarrow{C}$) communications between all source proxies and all source services $P(S_1) \ldots P(S_n)$, one-way communication ($\vec{C}$) between all source proxies $P(S_1) \ldots P(S_n)$ and the sink proxy $P(S_{FI})$, once received by the sink proxy round-trip communication ($\overleftrightarrow{C}$) takes place between the sink proxy $P(S_{FI})$ and the sink service S_{FI}, finally data are sent one-way ($\vec{C}$) back to the workflow engine E.

3.1.3 Fan-Out Pattern

This pattern is the reverse of the fan-in pattern, here the output from a single source is sent to multiple sinks. An initial source service is invoked with a control flow message (again no actual data are sent), the service returns data as output. These data are sent, asynchronously in parallel to multiple services as input, each service returns final data to the workflow engine.

For the standard orchestration model ((5), phase 5 of Fig. 5) $TC_{fo,nc}$ involves one-way communication ($\vec{C}$) between the source service S_{FO} and the workflow engine E, followed by round-trip communication ($\overleftrightarrow{C}$) between the workflow engine E and all sink services $S_1 \ldots S_n$.

For the Circulate architecture ((6), phase 6 of Fig. 5) $TC_{fo,circ}$ involves round-trip communication ($\overleftrightarrow{C}$) between the source proxy $P(S_{FO})$ and the source service S_{FO}, one-way communication ($\vec{C}$) between the source proxy $P(S_{FO})$ and all sink proxies $P(S_1) \ldots P(S_n)$, round-trip communication ($\overleftrightarrow{C}$) between all sink proxies $P(S_1) \ldots P(S_n)$ and all sink services $S_1 \ldots S_n$, finally one-way communication ($\vec{C}$)

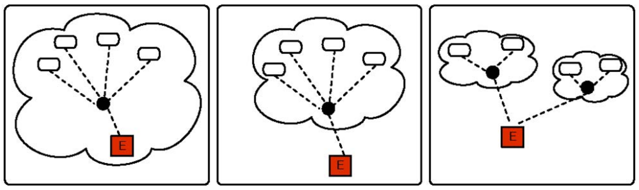


Fig. 6. Configurations from left to right: local LAN, remote LAN, Internet-scale. Services, proxies, and the workflow engine are always deployed on separate machines.

between all sink proxies $P(S_1) \dots P(S_n)$ and the workflow engine E.

4 Experimental Setup

A number of performance analysis experiments have been devised in order to observe and analyze the behavior of various workflow patterns and variables to compare the Circulate architecture with that of a standard centralized orchestration model. The following sections describe the structure of the resulting performance analysis experiments which are based the workflow patterns described in Section 3.

4.1 Node Size and Network Configurations

In our experimental set up, data flow with no data transformations (i.e., the output of one method invocation is the input to another). We use nonblocking asynchronous communication between the web services, although data forwarding between proxies occurs after all web services have finished execution.

For each of the three workflow patterns: sequence, fan-in, and fan-out, the time taken for the pattern to complete using centralized orchestration and using the Circulate architecture is recorded (in milliseconds) as the size of the input data (in Megabytes) is increased. This basic set up is then run incrementally over a number of workflow node sizes: 4, 8, and 16 nodes in order to explore the affects of scalability. Each node size (i.e., 4, 8, or 16) is then run over different network configurations to explore how network configuration affects the performance of a workflow.

We have selected the following network configurations (illustrated by Fig. 6) which mimic common scenarios when composing sets of geographically distributed services:

4.1.1 LAN Experiments

Moving the web services to a relatively uniform network topology in terms of speed, e.g., a LAN, allows for a simplified analysis of the two models. In a LAN, it is expected that the cost of the communication links $E \rightarrow S$, $E \rightarrow P$, $P \rightarrow S$, and $P \rightarrow P$ have little variance in terms of bandwidth. In this case the performance benefit with respect to the different workflow patterns may be exposed more readily. We explore this with a LAN configuration with a local workflow engine (same LAN) and a LAN configuration with a remote workflow engine (connected through a WAN).

4.1.2 Internet-Scale Experiments

PlanetLab [9] is a global research network of distributed servers that supports the development of new network services. We have heavily utilized this framework in order to evaluate the performance of the Circulate architecture across Internet-scale networks. We have subdivided the pattern-based PlanetLab experiments into three configurations which mimic the typical geographical distribution patterns found in workflow applications: "National" (all nodes in the same country), "Continental" (nodes in different countries on the same continent), and "Worldwide" (nodes spread throughout the globe). Finally, to demonstrate end-to-end performance, an end-to-end application is executed across the PlanetLab framework.

4.2 Experiment Cross Product

The configuration of our experiments mirror that of a typical workflow scenario, where collections of physically distributed services need to be composed into a higher level application, scaling from LAN to Internet-scale network configurations. In order to create a uniform test environment, proxies invoke web services with one operation which inputs and outputs Java byte arrays transferred using SOAP. The cross product of each workflow pattern, node size, and network configuration has been executed 100 times (combined standard deviation can be seen on graphs where a mean performance benefit is reported) over a set of distributed Linux machines running the WS-Circulate architecture discussed in Section 2.2. The experiments and graphs (throughout this paper) can be summarized as follows:

1. x-axis displays the size of the initial input file in Megabytes: for the sequence pattern this displays the size of data *sent* to the first service in the workflow, for fan-in and fan-out this represents the size of data *returned* by the first service.
2. For the LAN experiments the size of the initial input file ranges from 2 to 96 MB with a total of 12 data points collected. For the Internet-scale experiments the size of the initial input file ranges from 2 to 64 MB with a total of 10 data points collected. Each experiment is executed 100 times.
3. The y-axis displays the mean performance benefit. The mean performance benefit is calculated by dividing the mean time taken to execute the workflow using standard orchestration (i.e., nonproxy, fully centralized) and dividing it by the mean time taken to execute the workflow using the Circulate architecture. For example, a result of 2.0 means that the Circulate architecture executes a workflow twice as fast as standard orchestration.
4. Graphs that represent a performance benefit are shown together with a combined standard deviation of the two populations: orchestration and Circulate, represented by:

$$\sqrt{\left(\frac{stdev - non - circ}{mean - non - circ}\right)^2 + \left(\frac{stdev - circ}{mean - circ}\right)^2}. \qquad (7)$$

5 LAN Configuration

A pool of computers from the University of Edinburgh Distributed Informatics Computing Environment (DICE)[1]

1. http://www.dice.inf.ed.ac.uk.

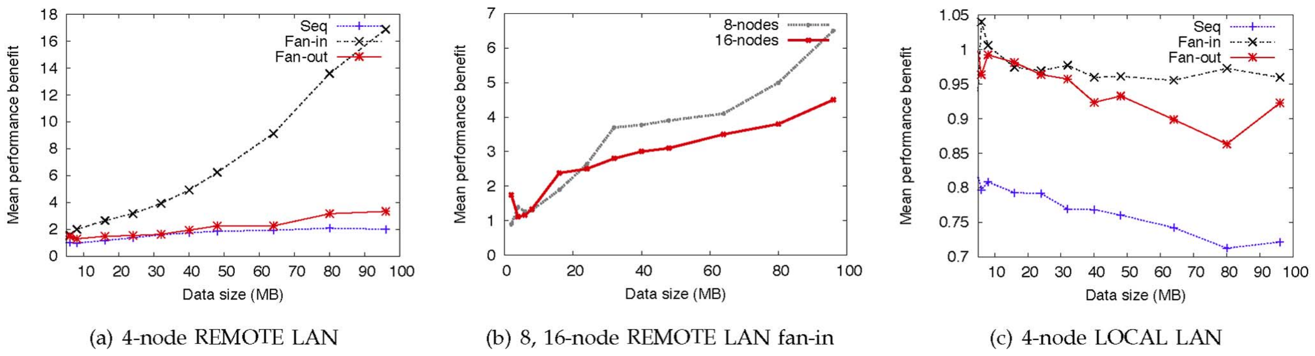


Fig. 7. Remote LAN configurations.

LAN were selected as workflow nodes for these configurations. These machines are all located in the same building and are connected to the network via a 100 Mbit network connection. All these machines share the same hardware/operating environment: Intel Core 2 Duo with 2 GB RAM running Ubuntu version 7.04. In addition to these matched machines, similar servers (separate machines) were chosen to act as the workflow engine, allowing us to explore the remote orchestration case and the local orchestration case.

5.1 LAN—Remote Orchestration

By moving the workflow engine outside of the LAN configuration, we can explore workflow engine to web service ($E \rightarrow S$) communication. This reflects a very common real-world scenario where a particular organization provides all of the web services used to compose a workflow, but these services are being orchestrated remotely. In particular, this is applicable to cloud computing scenarios (for example, Amazon Elastic Compute Clouds[2]) where services are typically hosted on groups of colocated machines and where the end user requiring these services is remote.

Fig. 7a displays the mean performance benefit across each of the workflow patterns, on a LAN with four nodes, where all web services are mapped to a single proxy and the workflow engine is remote from both proxies and services.

As one can observe from Fig. 7a, for all patterns the Circulate architecture outperforms the standard orchestration model, i.e., the mean performance benefit is larger than 1 even at low data sizes. The fan-in pattern shows the greatest benefit.

To justify, as the workflow engine lies outside the LAN where the web services and proxies are deployed, the $E \rightarrow S$ and $E \rightarrow P$ links can be considered "expensive" WAN hops. As the proxy is deployed on the same LAN as the web services it is invoking the $P \rightarrow S$ link is a less expensive LAN hop. Within a standard orchestration model, all data pass through a centralized workflow engine, hence all data are transferred over the expensive $E \rightarrow S$ link. Using the Circulate architecture, most of the processing for each of the patterns takes place over the less expensive $P \rightarrow S$ link; intermediate data are housed within the proxy. Data are only sent over the expensive $E \rightarrow P$ link at the start of the sequence pattern and the end of all patterns.

2. http://aws.amazon.com/ec2.

One can make the following observations from the LAN-scale experiments.

5.1.1 Patterns

With reference to Fig. 7a, the worst performing pattern in this scenario was the sequence pattern, however even this pattern demonstrates the significant advantage of using the Circulate architecture. At 2 MB, the mean performance benefit is just 1.03, at 96 MB this benefit increases to 2.0. Looking at the best performing pattern, fan-in, at 2 MB the mean performance benefit is 2.0, at 96 MB the mean performance benefit has risen to 16.8.

5.1.2 Node Size

In order to explore scalability, the same experiment was run over 8-node and 16-node configurations. One proxy is assigned to each group of four web services, resulting in two proxies for the 8-node experiment and four proxies for the 16-node experiment. As per the previous setup, the workflow engine is remote from all services and proxies. Fig. 7b illustrates the mean performance benefit of each of the node configurations when compared to the centralized orchestration model for the fan-in pattern. We have selected the fan-in pattern as it demonstrates the most consistent improvement in performance.

The Circulate architecture always outperforms the centralized orchestration model for each of the node configurations. To justify, as the number of proxies increases, so does the $P \rightarrow P$ communication, which adds a minimal additional cost to the execution of a workflow. However, this minimal overhead only adds an additional cost in comparison with using a configuration where all services share the same proxy. Our scaling experiment demonstrates that Circulate outperforms traditional orchestration for all node sizes in the remote LAN case.

5.1.3 Data Size

As the size of data involved in each of the patterns increases, the cost of processing the expensive WAN hops also increases. As the Circulate architecture reduces these more expensive WAN hops, the benefit of utilizing the architecture increases in proportion to the size of data involved in a workflow. To quantify (with reference to Fig. 7b), at 16 MB the 8-node fan-in pattern's mean performance benefit was 1.9, at 96 MB this increased to 6.4. At 16 MB the 16-node fan-in pattern's mean performance benefit was 2.2, at 96 MB this increased to 4.4.

TABLE 1
PlanetLab Node Selection

France	Germany	USA
inisa.fr	uni-konstanz.de	brown.edu
inria.fr	**uni-goettingen.de**	mit.edu
utt.fr	uni-paderborn.de	poly.edu
	fraunhofer.de	umd.edu
	tu-darmstadt.de	byu.edu
		postel.org
		iit-tech.net

5.2 LAN—Local Orchestration

In order to explore the limits of our approach, the workflow engine is deployed on a computer that is also connected to the proxies and web services via a network switch. This is a suitable experimental setup to test the assumption of communication link equality but does not necessarily reflect common deployed patterns of web services, normally the workflow engine is remote.

Fig. 7c displays the mean performance benefit as the data size increases across each of the three workflow patterns on a LAN running a local workflow engine, with 4-nodes, where four web services each share a proxy.

In a LAN environment we make the assumption that the cost of communication is relatively uniform, therefore the cost of $E \rightarrow S$, $E \rightarrow P$, $P \rightarrow S$, and $P \rightarrow P$ can be considered approximately equal. The Circulate architecture introduces extra communication links (between $P \rightarrow S$) across this uniform network topology, which in turn degrades the execution time of a workflow when compared to standard orchestration. With reference to Fig. 7c the fan-in pattern demonstrates similar performance to the standard orchestration model, while the standard orchestration model outperforms the fan-out and sequence patterns.

6 Internet-Scale Configurations

Moving the experiments to the PlanetLab configurations allows the Circulate architecture to be evaluated over Internet-scale networks. The PlanetLab environment configurations are based on the geographical location of the nodes, which in turn are used as an indicator of communication link cost. By grouping the nodes, certain realistic scenarios can be constructed for the experiments. For example, by using a group of nodes all located in France, one can execute a workflow simulating the interactions between collaborating French universities; such scenarios are common place in large-scale workflows. The PlanetLab configurations all use a remote workflow engine, this is a common feature of scientific workflows, as one can imagine a scientist in a remote location orchestrating resources from a number of collaborating institutions.

Due to the variable nature of PlanetLab, node selection was a complex matter. We ran extensive tests to locate groups of nodes that we could reliably access throughout the duration of our experiments. After these preliminary tests nodes from France, Germany, and the USA were selected, each nodes are shown in Table 1. Each workflow pattern was then executed over nodes from Table 1, mashed up using different geographical network configurations.

TABLE 2
National (4-Node), Continental (8-Node), and World Wide (16-Node) PlanetLab Mean Performance Benefits

Configuration	Best	Worst	Mean
4-node – France			
Seq	1.49	1.27	1.39
Fan-in	1.30	1.08	1.27
Fan-out	1.61	1.4	1.43
Overall			1.36
4-node – Germany			
Seq	1.72	1.50	1.58
Fan-in	3.13	2.72	2.94
Fan-out	1.82	1.64	1.69
Overall			2.07
4-node – USA			
Seq	1.76	1.50	1.61
Fan-in	1.95	1.69	1.85
Fan-out	1.85	1.30	1.61
Overall			1.69
8-node – Europe			
Seq	2.42	2.18	2.27
Fan-in	1.69	1.56	1.61
Fan-out	1.87	1.67	1.72
Overall			1.87
8-node – USA			
Seq	2.04	1.64	1.75
Fan-in	2.22	1.79	2.04
Fan-out	1.27	1.11	1.18
Overall			1.66
16-node – World wide			
Seq	1.44	1.12	1.29
Fan-in	2.63	1.92	2.32
Fan-out	1.89	1.04	1.22
Overall			1.61

The breakdown of each pattern, network configuration, and mean performance benefit is displayed in Table 2

6.1 National-4-Node

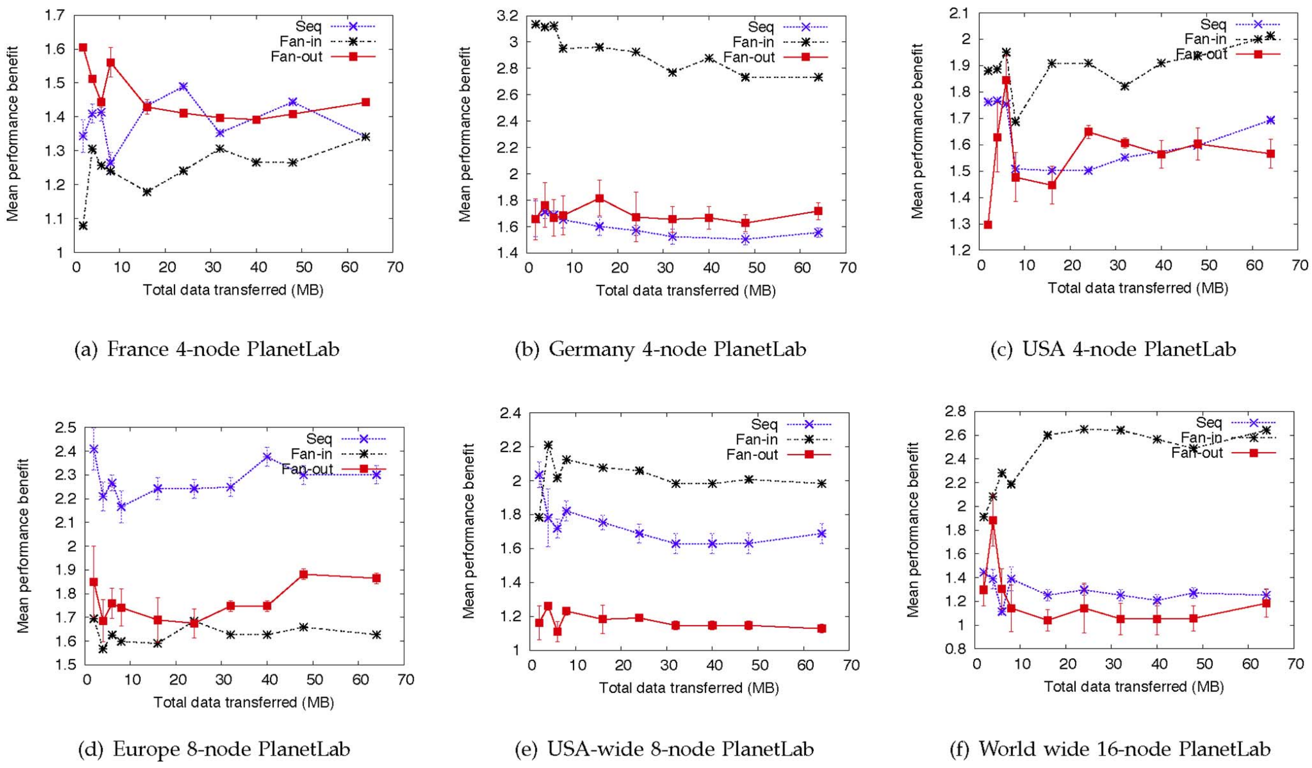
Within the national configuration, four nodes were selected to deploy the web services, a further node acted as a shared proxy and the workflow engine was deployed on a final node that is a separate node from the proxy and services. The mean performance benefit for the France configuration is displayed in Fig. 8a, the Germany configuration in Fig. 8b, and the USA configuration in Fig. 8c.

6.2 Continental-8-Node

In order to separate the nodes further, each workflow pattern was executed across two continental configurations. The first was a European set, which contained nodes from both France and Germany. The second utilized nodes across the USA. Eight web services were deployed across eight nodes, a further two nodes acted as proxies for groups of four web services, a final node acted as a workflow engine that is a separate node from any of the proxies and services. The mean performance benefit for the European configuration is displayed in Fig. 8d and USA-wide configuration in Fig. 8e.

6.3 World Wide-16-Node

The final PlanetLab experiment executed each pattern across all available nodes in the previous configurations. Sixteen web services were deployed across 16 nodes, four proxies were deployed across separate nodes, which manage four web services each, a final node acted as the workflow engine that is a separate node from any of the proxies and services.



(a) France 4-node PlanetLab (b) Germany 4-node PlanetLab (c) USA 4-node PlanetLab

(d) Europe 8-node PlanetLab (e) USA-wide 8-node PlanetLab (f) World wide 16-node PlanetLab

Fig. 8. National (4-node), continental (8-node), and global (16-node) PlanetLab configurations.

The mean performance benefit for the world wide configuration is displayed in Fig. 8f.

One can make the following observations from the Internet-scale experiments:

The Circulate architecture always outperformed the centralized orchestration model. The greatest overall improvements in performance were seen on the 4-node Germany configuration (2.07 overall mean performance benefit), 8-node Europe configuration (1.87 overall mean performance benefit), and 4-node USA configuration (1.69 overall mean performance benefit).

6.4 Patterns

If we calculate the mean across all Internet-scale network configurations (i.e., the four, eight, and 16-node) from Table 2 the following trends can be observed: the sequence pattern demonstrated a performance benefit of 1.65, fan-in a performance benefit of 2.01 and fan-out a performance benefit of 1.48.

6.5 Data Size

As one can observe from Figs. 8a, 8b, 8c, 8d, 8e, and 8f, the Internet-scale configurations were more variable than the LAN experiments. However, the general trend was that as the data size increased the benefit either improved or remained relatively constant.

6.6 Node Size

Obtaining general trends on the PlanetLab results was not as straight forward as the LAN configurations. Unlike the relatively uniform LAN environment, the performance across PlanetLab is heavily dependent on the quality of links utilized and current load of the network (observed by the error bars in Fig. 8). As we have discovered PlanetLab links vary radically in quality. For example, the overall improvement in the 4-node German configuration (2.07 mean performance benefit) is higher than in the 4-node France configuration (1.36 mean performance benefit). Analyzing individual runs it was found that the $E \rightarrow S$ and $E \rightarrow P$ communication cost was higher in the German nodes. As the Circulate architecture reduces these expensive links, the mean performance benefit across pattern was lower.

For standard orchestration the quality of the $E \rightarrow S$ links is the overall factor affecting performance, for Circulate it is the quality of the $E \rightarrow P$ and $\rightarrow P$ links. Therefore, one has to place proxies carefully in order to ensure that the $P \rightarrow P$ links are faster than the $E \rightarrow S$ links, these optimization and proxy placement strategies are left to future work.

Most promising, however, was that in all cases, when all workflow patterns were taken into account, the Circulate architecture always speedup the execution time of a workflow. This is important as it demonstrates that even if one pattern is misbehaving, a workflow, which will be composed of an arbitrary number of patterns will still experience performance improvements.

7 End-To-End Application: Montage

Although the focus of our paper has primarily been on pattern-based performance analysis, it is important to demonstrate the Circulate architecture on an end-to-end application. Fig. 9 illustrates the Montage workflow, a benchmark in the High Performance Computing community and representative of a class of large-scale, data-intensive workflows. Montage constructs custom "science-grade"

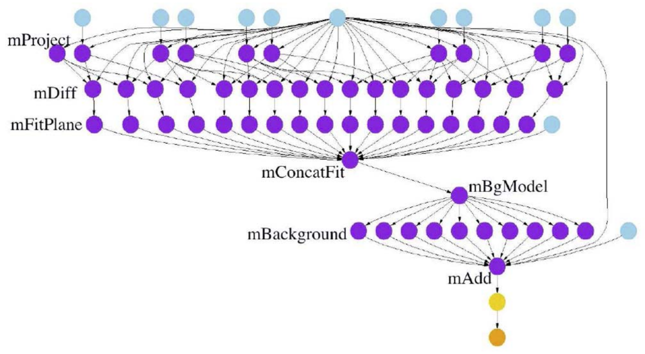


Fig. 9. Montage use-case scenario.

astronomical image mosaics from a set of input image samples. It illustrates several features of data-intensive scientific workflows and it can result in large data flow requirements, intermediate data can be three times the size of input data.

The Montage workflow is represented as a DAG, each component of the DAG (along with corresponding input-output data ratios) is explained as follows:

1. mProject: reprojects an image to the coordinate system defined in a header file. $(\text{output} = \text{input})$
2. mDiff/mFitPlane: three inputs (one header and two images) *fan-in* to the mDiff function, which finds the difference between the two images, the output is then passed through the mFitPlane function (usually executed on the same machine), which fits a plane to the difference image. $\text{Output} = 15\text{-}20\%$ of a typical image for each image triplet.
3. mConcatFit: a simple concatenation of the plane fit parameters from multiple mDiff/mFitPlane jobs into a single file. *fan-in* pattern with 18 inputs (from different resources), which are passed through the mConcatFit function.
4. mBgModel: models the sky background using the plane fit parameters from mDiff/mFitPlane and computes planar corrections for the input images that will rectify the background across the entire mosaic. mBgModel is a *fan-out* pattern, where the output is distributed to 10 sinks.
5. mBackground/mAdd: the mBackground function rectifies the background in a single image, $\text{output} = \text{input}$. Data from each mBackground computation are sent to the mAdd function, which coadds a set of reprojected images to produce a mosaic as specified in a template header file. This forms a *fan-in* pattern with 10 inputs to the mAdd function $(\text{output} = 70\text{-}90\%$ the size of inputs put together).

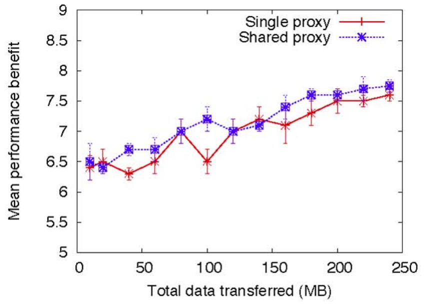
The Montage application (web services and data) along with a set of proxies were deployed on PlanetLab nodes spanning the USA. Our deployment maintains the number of services (i.e., fan-ins and fan-outs), data sizes, and importantly the input-output relationships of Montage.

Two experiments were performed: first, "single proxy" where each web service is maintained by its own proxy, and "shared proxy" where groups of four web services were maintained by a single proxy. Proxies and services were scattered across PlanetLab nodes spanning the USA, proxies were always deployed on a separate machine within the same domain as the web service it is invoking and the workflow engine was always remote from both proxy and web service. This deployment was executed 50 times across the PlanetLab framework. As with previous experiments the x-axis represents the size of the input file and 95 percent confidence intervals are provided for every mean performance benefit. A total of 13 data points were collected from 10 to 240 MB. As we are focusing purely on optimizing data transfer, processing times in both models have not been taken into account as these remain the same regardless of the number of services served per proxy.

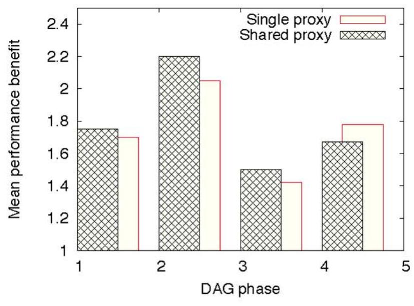
Fig. 10a illustrates the end-to-end mean performance benefit of the Montage application using Circulate and standard centralized orchestration. Fig. 10b illustrates the mean performance benefit (average across 13 data points and 50 runs) per phase of the Montage application and demonstrates how each phase collectively provides an end-to-end benefit. Phase 1 represents mProject to mFitPlane, phase 2 mConcatFit, phase 3 mBgModel, and phase 4 mBackground to mAdd.

The end-to-end Montage application resulted in a mean performance benefit of 6.95 for the single proxy configuration over all 13 data volumes tested.

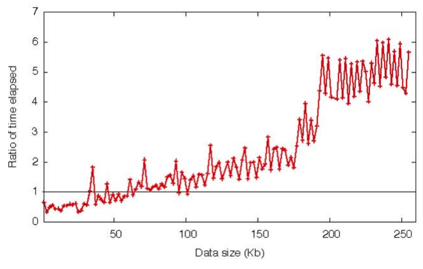
This benefit increases slightly as the input data size increased: from an average of 6.4 (30 seconds for orchestration, 192 seconds for Circulate) at 10 MB to an average of 7.6 (246 seconds for orchestration, 1,869.6 for Circulate) at 240 MB. The shared proxy configuration resulted in a mean performance benefit of 7.12, again the benefit increased



(a) Montage end-to-end performance



(b) Montage DAG phase performance



(c) The overhead of invoking a proxy

Fig. 10. (a) Montage end-to-end performance, (b) Montage individual phases performance, and (c) proxy overhead.

slightly as the data size increased: from 6.5 (24 seconds for orchestration, 156 seconds for Circulate) at 10 MB to 7.75 (216 seconds for orchestration, 1,674 seconds for Circulate) at 240 MB. The shared proxy resulted in a marginal benefit over the single proxy configuration due to reduced data transfer to and from proxies. With reference to the pattern-based performance analysis, we confirm that the benefit of using the Circulate architecture increases when isolated patterns are placed together to form a larger application.

7.1 Proxy Overhead

Fig. 10c displays the average time (as a ratio: nonproxy centralized time divided by Circulate elapsed time) it takes to make a single invocation to a vanilla web service and obtain the result versus an invocation to a proxy that invokes the service on the orchestration engines behalf and returns a reference to its data. The workflow engine is remote to both proxy and service. Results under the horizontal line indicate the vanilla approach is optimal, results over the line show a benefit of using the Circulate architecture.

Circulate effectively separates out the control flow and data flow messages from one another; the workflow engine invokes proxies with smaller control flow messages and proxies pass larger data flow messages directly to one another. Although workflow languages such as BPEL do not explicitly separate out these two types of messages it is useful to think of data transfer in this way. The proxy overhead experiment demonstrates that Circulate is only suited to workflows where large quantities of intermediate data flow between services in a workflow. From the results we conclude that due to the overhead of the proxy, when dealing with input data sizes of less than $\sim$100K of data the Circulate architecture offers no performance benefit to web services. Anything over $\sim$100K of data the proxy begins to speedup the execution time of the invocation due to the storage of the results within the proxy.

8 Related Work

This section discusses all related work from the literature, spanning pure choreography languages, enhancements to widely used modeling techniques, i.e., BPMN, decentralized orchestration, data flow optimization architectures, and Grid toolkits.

8.1 Choreography Languages

For completeness it is important to discuss the current state of the art in choreography techniques. There are an overwhelming number of pure orchestration languages. However, relatively few targeted specifically at choreography, the most prevalent being WS-CDL [5], BPEL4Chor [12], and Let's Dance [30].

There are even fewer complete implementations of choreography languages, this means that choreography techniques are rarely deployed in practice. For example, there are only two documented prototype implementations of the WS-CDL specification. WS-CDL+, an extended specification [18] has been implemented in prototype form, although only one version, version 0.1 has been released. A further partial implementation [14] of the WS-CDL specification is currently in the prototype phase. The other widely known implementation is pi4soa,[3] an Eclipse plug-in that provides a graphical editor to compose WS-CDL choreographies and generate from them compliant BPEL. Maestro [11] is an implementation of the Let's Dance language and supports the static analysis of global models, the generation of local models from global ones, and the interactive simulation (not enactment) of both local and global modes. In the BPEL4Chor space, a web-based editor[4] allows engineers to graphically build choreography models.

8.2 Techniques in Data Transfer Optimization

There are a limited number of research papers that have identified the problem of a centralized approach to service orchestration.

The *Flow-based Infrastructure for Composing Autonomous Services* or FICAS [23] is a distributed data-flow architecture for composing software services. Composition of the services in the FICAS architecture is specified using the Compositional Language for Autonomous Services (CLAS), which is essentially a sequential specification of the relationships among collaborating services. This CLAS program is then translated by the build-time environment into a control sequence that can be executed by the FICAS runtime environment.

FICAS is intrusive to the application code as each application that is to be deployed needs to be wrapped with a FICAS interface. In contrast, our proxy approach is more flexible as the services themselves require no alteration and do not even need to know that they are interacting with a proxy. Furthermore, our proxy approach introduces the concept of passing references to data around and deals with modern workflow standards.

Service Invocation Triggers [7], or simply Triggers are also a response to the bottleneck problem caused by centralized workflow engines. Triggers collect the required input data before they invoke a service, forwarding the results directly to where these data are required. For this decentralized execution to take place, a workflow must be deconstructed into sequential fragments that contain neither loops nor conditionals and the data dependencies must be encoded within the triggers themselves. This is a rigid and limiting solution and is a barrier to entry for the use of proxy technology. In contrast with our proxy approach, because data references are passed around, nothing in the workflow has to be deconstructed or altered, which means standard orchestration languages such as BPEL can be used to coordinate the proxies.

In [24] a similar (pure choreography) approach is also proposed. Authors introduce a methodology for transforming the orchestration logic in BPEL into a set of individual activities that coordinate themselves by passing tokens over shared, distributed tuple spaces. The model suitable for execution is called Executable Workflow Networks (EWFN), a Petri nets dialect. This approach utilizes a pure choreography model which has many additional modeling and enactment problems associated with it, due primarily

3. http://sourceforge.net/projects/pi4soa.
4. http://www.bpel4chor.org/editor.

to the complexity of message passing between distributed, concurrent processes.

Data caching techniques in Grid workflows are proposed in [8]. This architecture caches "virtual data" of previous queries, so any overlapping queries and processing do not have to be repeated. The Circulate architecture stores data at a proxy so that it can be transferred directly to the next stage of the workflow, avoiding costly network hops back to the workflow engine. It is then up to the user to clean up stored data afterwards, by using the methods discussed in Section 2.2. The automated data caching techniques proposed could be applied to the Circulate proxies to further enhance performance.

8.3 Third-Party Data Transfers

This paper focuses primarily on optimizing service-oriented workflows, where services are: not equipped to handle third-party transfers, owned and maintained by different organizations, and cannot be altered in anyway prior to enactment. For completeness it is important to discuss engines that support third-party transfers between nodes in task-based workflows.

Directed Acyclic Graph Manager (DAGMan) [10] submits jobs represented as a DAG to a Condor pool of resources. DAGMan removes the workflow bottleneck as data are transferred directly between vertices in a DAG, however, focuses purely on Condor bases Grid jobs and does not address the bottleneck problems associated with orchestrating service-oriented workflows.

Triana [27] is an open-source problem solving environment. It is designed to define, process, analyze, manage, execute, and monitor workflows. Triana can distribute sections of a workflow to remote machines through a connected peer-to-peer network.

OGSA-DAI [19] is a middleware product that supports the exposure of data resources on to Grids. This middleware facilitates data streaming between local OGSA-DAI instances. Our architecture could be implemented on this platform to take advantages of its streaming model.

Grid Services Flow Language (GSFL) [21] addresses some of the issues discussed in this paper in the context of Grid services, in particular services adopt a peer-to-peer data flow model. However, individual services have to be altered prior to enactment, which is an invasive and custom solution, something that is avoided in the Circulate architecture.

9 Conclusions

As the number of services and the size of data involved in workflows increases, centralized orchestration techniques are reaching the limits of scalability. In standard orchestration: all data passes through a centralized engine resulting in unnecessary data transfer, wasted bandwidth and the engine to become a bottleneck to the execution of a workflow. Decentralized choreography techniques, although optimal in terms of data transfer are far more complex to build due to message passing between distributed, concurrent process and in practice and rarely deployed.

This paper has presented the Circulate architecture; a centralized orchestration model of control with a peer-to-peer choreography model of data transfer. This extra functionality is achieved through the deployment of a lightweight proxy that provides a gateway and standard API to web service invocation. Importantly, proxies can be deployed without disrupting current services and with minimal changes in the workflows that make use of them. This flexibility allows a gradual change of infrastructures, where one could concentrate first on improving data transfers between services that handle large amounts data.

An open-source web services implementation, WS-Circulate, served as the platform for our evaluation across LAN and Internet-scale configurations through the PlanetLab network. Through our experimentation we have demonstrated that by reducing data transfer, the Circulate architecture significantly speeds up the execution time of workflows across common workflow patterns and network topologies. Moreover, it scales with data size and common workflow topologies, and consistently outperforms centralized orchestration techniques. The Montage DAG demonstrated that the benefit of using the Circulate architecture increases when isolated patterns are placed together to form a larger application.

Future work includes the following challenges:

- *Architecture Evolution.* Although this paper has discussed a web services-based implementation, Circulate is a general architecture and mappings could be provide to multiple back end technologies, e.g., Condor [22]. This would allow multiple technology sets to be optimized and orchestrated via a standard workßow language and workflow engine.
- *Compressing Web Service Content.* Techniques, e.g., gzipped SOAP[5] have been proposed to reduce the quantity of data transferred between web services. We plan to integrate such techniques into the Circulate architecture.
- *Proxy Placement.* This architecture also opens up a rich set of additional optimizations with respect to dynamic proxy deployment, i.e., load balancing depending on network traffic.
- *Data Transformation.* A common workflow task is to change the output data from one service into a slightly different format to use as input into another service, this process is known as Shimming. This will affect the performance of a workflow further as it introduces an extra service into the workflow chain. If a shim is implemented as a custom transformation web service this step will simply be included in the workflow specification. Future work includes implementing a shim loader component, where custom shims can be uploaded to the proxy, allowing shims to be performed locally at a proxy, avoiding the introduction of further network hops.

References

[1] W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns," *Distributed and Parallel Databases,* vol. 14, pp. 5-51, 2003.

[2] Apache Axis. http://ws.apache.org/axis, 2011.

5. http://www.ibm.com/developerworks/webservices/library/ws-sqzsoap.html.

[3] A. Barker and J. van Hemert, "Scientific Workflow: A Survey and Research Directions," *Proc. Seventh Int'l Conf. Parallel Processing and Applied Math.*, R. Wyrzykowski et al., ed., pp. 746-753, 2008.

[4] A. Barker, J.B. Weissman, and J. van Hemert, "Orchestrating Data-Centric Workflows," *Proc. IEEE Eighth Int'l Symp. Cluster Computing and the Grid (CCGrid)*, pp. 210-217, 2008.

[5] A. Barros, M. Dumas, and P. Oaks, "A Critical Overview of the Web Services Choreography Description Language (WS-CDL)," BPTrends Newsletter 3, 2005.

[6] A. Barros, M. Dumas, and A. ter Hofstede, "Service Interaction Patterns," *Proc. Third Int'l Conf. Business Process Management*, pp. 302-318, 2005.

[7] W. Binder, I. Constantinescu, and B. Faltings, "Decentralized Ochestration of Composite Web Services," *Proc. Int'l Conf. Web Services (ICWS '06)*, pp. 869-876, 2006.

[8] D. Chiu and G. Agrawal, "Hierarchical Caches for Grid Workflows," *Proc. IEEE/ACM Ninth Int'l Symp. Cluster Computing and the Grid (CCGRID '09)*, pp. 228-235, 2009.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An Overlay Testbed for Broad-Coverage Services," *SIGCOMM Computer Comm. Rev.*, vol. 33, no. 3, pp. 3-12, 2003.

[10] Condor Team, http://www.cs.wisc.edu/condor/dagman, 2010.

[11] G. Decker, M. Kirov, J.Maria Zaha, and M. Dumas, "Maestro for Let's Dance: An Environment for Modeling Service Interactions," *Proc. Session of the Fourth Int'l Conf. Business Process Management (BPM)*, 2006.

[12] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," *Proc. IEEE Int'l Conf. Web Services (ICWS '07)*, pp. 296-303, 2007.

[13] E. Deelman, "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example," *Proc. IEEE Second Int'l Conf. e-Science and Grid Computing*, 2006.

[14] L. Fredlund, "Implementing WS-CDL," *Proc. Second Spanish Workshop Web Technologies (JSWEB '06)*, 2006.

[15] A.J.G. Hey and A.E. Trefethen, "The Data Deluge: An e-Science Perspective," *Grid Computing—Making the Global Infrastructure a Reality*, pp. 809-824, Wiley and Sons, 2003.

[16] D. Hollingsworth, *The Workflow Reference Model*, Workflow Management Coalition, 1995.

[17] J.C. Jacob et al., "The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets," *Proc. Earth Science Technology Conf.*, June 2004.

[18] Z. Kang, H. Wang, and P.C. Hung, "WS-CDL+: An Extended WS-CDL Execution Engine for Web Service Collaboration," *Proc. IEEE Int'l Conf. Web Services (ICWS '07)*, pp. 928-935, 2007.

[19] K. Karasavvas, M. Antonioletti, M. Atkinson, N.C. Hong, T. Sugden, A. Hume, M. Jackson, A. Krause, and C. Palansuriya, "Introduction to OGSA-DAI Services," *Proc. First Int'l Conf. Scientific Applications of Grid Computing*, pp. 1-12, 2005.

[20] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon, "Web Services Choreography Description Language (WS-CDL) Version 1.0," W3C Candidate Recommendation, 2005.

[21] S. Krishnan, P. Wagstrom, and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services," technical report, Argonne Nat'l Argonne Nat'l Laboratory, 2002.

[22] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," *Proc. Eighth Int'l Conf. Distributed Computing Systems*, June 1988.

[23] D. Liu, K.H. Law, and G. Wiederhold, "Data-Flow Distribution in FICAS Service Composition Infrastructure." *Proc. 15th Int'l Conf. Parallel and Distributed Computing Systems*, 2002.

[24] D. Martin, D. Wutke, and F. Leymann, "A Novel Approach to Decentralized Workflow Enactment," *Proc. IEEE 12th Int'l Conf. Enterprise Distributed Object Computing (EDOC '08)*, pp. 127-136, 2008.

[25] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows," *Bioinformatics*, vol. 20, pp. 3045-3054, 2004.

[26] D. Sulakhe, A. Rodriguez, M. Wilde, I.T. Foster, and N. Maltsev, "Interoperability of GADU in Using Heterogeneous Grid Resources for Bioinformatics Applications," *IEEE Trans. Information Technology in Biomedicine*, vol. 12, no. 2, pp. 241-246, Mar. 2008.

[27] I. Taylor, M. Shields, I. Wang, and R. Philp, "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS '03)*, pp. 16-27, 2003.

[28] *Workflows for e-Science: Scientific Workflows for Grids*, I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, eds. Springer-Verlag, 2006.

[29] The OASIS Committee. Web Services Business Process Execution Language (WS-BPEL) Version 2.0, 2007.

[30] J.M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede, "Let's Dance: A Language for Service Behavior Modelling," *Proc. Confederated Int'l Conf. Move to Meaningful Internet Systems (OTM)*, R. Meersman and T.Z., eds., pp. 145-162, 2006.

Adam Barker received the PhD degree from the School of Informatics, University of Edinburgh, in 2007. He is a lecturer (assistant professor) in computer science at the University of St. Andrews, United Kingdom. Prior to obtaining a faculty position, he worked as a postdoctoral researcher at the University of Melbourne, the University of Oxford, and the National e-Science Centre (NeSC), University of Edinburgh. His broad research interests concentrate on the theoretical foundations and effective engineering of large-scale distributed systems. His research agenda is pursued by building and evaluating novel architectures and algorithms based on sound foundations in systems research. For a more detailed biography and list of publications, please visit http://www.adambarker.org.

Jon B. Weissman received the BS degree from Carnegie-Mellon University in 1984, and the MS and PhD degrees from the University of Virginia in 1989 and 1995, respectively, all in computer science. He is a leading researcher in the area of cloud and grid computing. His involvement dates back to the influential Legion project at the University of Virginia during his PhD degree. He is currently an associate professor of Computer Science at the University of Minnesota where he leads the Distributed Computing Systems Group. His current research interests include Grid computing, distributed systems, high-performance computing, resource management, reliability, and e-science applications. He works primarily at the boundary between applications and systems. Being a visitor at the Institute, he has been appointed an Honorary Fellow of the College of Science and Engineering at the University of Edinburgh. He is a senior member of the IEEE.

Jano I. van Hemert received the PhD degree in mathematics and physical sciences from the Leiden University, The Netherlands, in 2002. He is a project manager and academic liaison at Optosan innovative retinal imaging company with a vision to be recognized as the leading provider of retinal diagnostics. He has held research positions at the University of Edinburgh (United Kingdom), Leiden University (NL), the Vienna University of Technology (AT), and the National Research Institute for Mathematics and Computer Science (NL). In 2004, he was awarded the Talented Young Researcher Fellowship by the Netherlands Organization for Scientific Research. In 2009, he was recognized as a promising young research leader with a Scottish Crucible. In September 2010, he became an Honorary Fellow of the College of Science and Engineering at the University of Edinburgh in recognition of his work for the School of Informatics.