Analysis of Integration Models for Service Composition

David Liu
Dept. of Electrical Engineering
Stanford University
Stanford, CA 94305, USA
davidliu@stanford.edu

Kincho H. Law
Dept. of Civil & Environmental Engineering
Stanford University
Stanford, CA 94305, USA
law@cive.stanford.edu

Gio Wiederhold
Computer Science Department
Stanford University
Stanford, CA 94305, USA
gio@db.stanford.edu

ABSTRACT

This paper studies service integration infrastructures that support the execution of megaservices – large-scale applications that are composed of autonomous service modules. Integration infrastructures are classified according to their control-flow and data-flow structures. We analyze the effects of data-flows on the performances of the centralized and distributed data-flow models. A mathematical model is built to compare the performances of megaservices. Particularly, aggregated cost and response time metrics are defined and evaluated. We arrive at the conclusion that the distributed data-flow model is in general superior in performance. We also identify the key system parameters as well as system bottlenecks. The analysis provides recommendations for a few techniques to build high-performance and scalable service integration infrastructures based on the distribution of data-flows.

Keywords

Service composition, performance evaluation, distributed data-flow, service integration infrastructure, FICAS

1. INTRODUCTION

1.1. Background

A software engineering paradigm has long been envisioned where large applications are decomposed into cooperating components [9]. Such vision is echoed in the megaprogramming framework [1, 17], which builds on software components called megamodules that capture the functionality of services provided by large organizational units. Megamodules are independently maintained and provide their clients with services specified by their interface and encapsulate local structures that implement their services [11]. Megamodules are linked together according to composition specifications [15] to form megaservices.

With the rapid development of Internet and network technologies, the foundation to realize such a vision of software composition exists. The computing world is evolving toward an interconnected web of autonomous services that are managed under the providers' administrative domains. Autonomous services are typically computational or data intensive. The Internet provides a wide variety of services, although they are rarely envisaged for composition. Examples include travel reservation, book purchasing, weather services, financial data summaries, and newsgathering. Other services include simulation programs [16], engineering, logistics, and business services.

In service composition, a systematic integration framework is required to incorporate transmission protocols for control messages and data traffic so that autonomous services can be coordinated to perform a task. A service integration infrastructure ties together distributed computing resources to form a coherent computing environment. Specifically, integration infrastructures provide support for execution of various autonomous services, scheduling and coordination of the services, data communication among the services, and exception handlings, etc.

1.2. Objectives

The focus of this paper is on analyzing the effects of data-flows on the performance of various service integration infrastructures. Much research effort has been devoted in the past decades to control-flow scheduling in order to enhance the performance of distributed services. Less attention was given to data-flow based optimization techniques. The objectives of this paper are to point out the significance of data-flows on performance and to provide mathematical basis for developing service integration infrastructures that utilize distribution of data-flows.

We start our analysis by classifying service integration infrastructures into four conceptual models. The classification is based on how controls and data are communicated among autonomous services and megaservices. The computing environment is then modeled as a set of processors interconnected by a completely connected communication network, which can represent most modern computing environments. A mathematical model is built to characterize the performance metrics of individual system components, based on which aggregated cost and response time for executing megaservices can be evaluated.

Given an abstract mathematical representation for the integration models, we analyze and compare in detail the performances of the centralized data-flow model and the distributed data-flow model. By tuning the values of the underlying system components, we can study how the overall system responds to different settings of hardware platform. The results provide valuable information toward designing high-performance and scalable service integration infrastructure based on distribution of data-flows.

2. SYSTEM OVERVIEW

2.1. Service Integration Models

Conceptually, a distributed computing environment is viewed as a set of processors interconnected by a communication network. We characterize the work performed by megaservice and autonomous services in terms of either computation or communication. For computation, local processing is conducted on a single processor, and it involves no interaction between multiple processors. For communication, messages are passed between two processors. We do not include special support for broadcast and multicast, which can be modeled as multiple pair-wise messages. There are two types of messages: control messages and data messages, distinguished by their use at the recipients of the messages. Control messages are mostly short messages that are used to trigger state changes at the receiving services. Data messages are mostly large data contents that are given to the receiving services for processing. Examples of control message include service invocation requests and status polling requests. Examples of data messages include engineering design data and weather information to conduct simulation.

To execute a megaservice, control and data messages need to be exchanged among autonomous services. We use control-flow to describe the set of partially ordered control messages, and use data-flow to describe the set of partially ordered data messages. Service integration infrastructures differ in how control-flows and data-flows are formed and managed. As shown in Figure 1, service integration infrastructures are categorized into four models:

- Centralized control-flow, centralized data-flow (1C1D)
- Centralized control-flow, distributed data-flow (1CnD)
- Distributed control-flow, centralized data-flow (nC1D)
- Distributed control-flow, distributed data-flow (nCnD)

The 1C1D model has the simplest structure. The megaservice is the central exchange point for both control and data messages. It naturally fits client-server architectures, where autonomous services act as servers and the megaservice functions as the client. Data and requests are often passed together from the megaservice to a desired autonomous service, and the results are returned to the megaservice for further processing. When additional functionalities are needed from other autonomous services, data and requests are again sent out from the megaservice.

Due to its simple conceptual model and easy implementation, the 1C1D model is the most common model for current service integration infrastructures. However, megaservices become communication bottlenecks in the 1C1D model. The centralized communication topology makes the 1C1D model unscalable. It is especially problematic in an Internet environment, where the communication links between the megaservice and autonomous services are likely to be of limited bandwidth. On the other hand, the high-speed networks deployed between autonomous services will not be utilized under the 1C1D model.

The 1CnD model can alleviate the deficiencies in the 1C1D model. The 1CnD model maintains the same centralized control mechanism as the 1C1D model. However, the improvement come from the scheme that data can be passed directly between autonomous services without going through the megaservice. Data communications among autonomous services are introduced, resulting in distributed data-flows. The megaservice does not need to function as an intermediate node on the data-flow path when data are exchanged between two autonomous services. The megaservice can simply instruct two autonomous services to establish a data-flow through which data can be directly communicated. For example, as illustrated in Figure 1(b), suppose the megaservice *M* needs *S2* to process some data generated by *S1*. Rather than fetch the data from *S1* and then pass the data onto *S2*, *M* can inform *S2* to fetch data directly from *S1*.

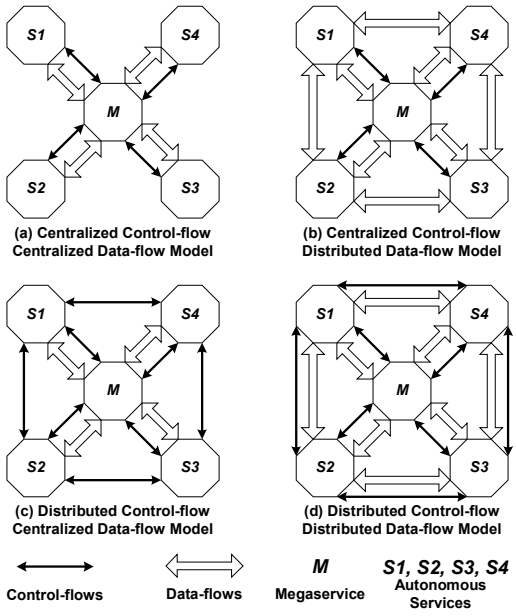

Figure 1: Service Integration Models

Two integration models with distributed control-flow are included to complete the classification. The nC1D model is a variation of the 1C1D model with distributed control-flow, and the nCnD model is an extension of the 1CnD model. In the distributed control-flow models, control messages can be sent between autonomous services, and the course of megaservice execution is coordinated by multiple autonomous services. A good example of distributed control-flow model can be found in data-flow computer architectures [3, 5, 13] where the execution of a program is partially controlled by the flow of data rather than successive fetching of instructions. A parallel program is compiled into operational code segments that are distributed to distinctive functional units, and the presence of operands activates the execution of the code segments. Given its ability to exploit the natural parallelism of algorithms [4], data-flow architecture has been seen as a promising approach in designing high performance multi-processor machines.

However, there are difficulties in effectively applying distributed control-flow models to conduct service composition. Because operational code segments need to be distributed to relevant function units for execution, the distributed control-flow models would require homogeneity in the underlying hardware platform. This requirement may easily be met in building parallel computers, but not in constructing heterogeneous service composition infrastructure. Also, it remains a technical challenge to convert a centralized specification of control sequences that a megaservice uses into operational code segments that can be effectively used to execute the megaservice. Due to these limitations, distributed control-flow models have been adopted only for special-purpose applications, where code segments are installed on individual functional units and a distributed

application environment is constructed from bottom up. The topics of exploiting distribute control-flow models are beyond the scope of this paper. We will focus our analysis on centralized control-flow models.

We have argued intuitively that the distribution of data-flows among an integration infrastructure would have significant impact on the system performance. For the rest of this paper, we will formalize our observation and conduct comparison between the 1C1D model and the 1CnD model.

2.2. System Modeling

In order to evaluate the performance of megaservices under different integration models, we need to first characterize and give mathematical definition to the components within the computing environment, including the hardware platform, the autonomous services, and the megaservices.

As illustrated in Figure 2, the hardware platform is modeled as a set of processor nodes $\boldsymbol{P} = \{P_0, P_1, \ldots, P_n\}$ tied together by a completely connected network. Associated with every processor P_i is the processor capacity CP_i expressed in terms of number of cycles that the processor can handle in unit time. Furthermore, associated with each pair of processors (P_i, P_j) is the communication capacity CM_{ij} expressed in terms of the volume of data that can be transmitted from processor P_i to processor P_j in unit time. A communication channel originated from a processor P_i to itself can also exist, with capacity CM_{ii}.

The communication network is modeled as a set of point-to-point links that connect every processor with each other. Each communication link operates independent of each other, as there is no shared medium between links. It is a simplified model for most of the real-world network architectures, but a sufficient approximation for the purpose of the analysis here. More complex models can be built by introducing additional constraints on the communication capacities.

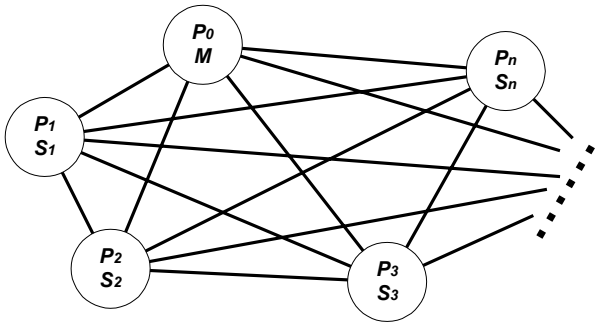

Figure 2: Integration System Modeling

$\boldsymbol{S} = \{S_1, \ldots, S_n\}$ is a set of autonomous services, each performing some specific operations. Conceptually, autonomous service S_i runs on the local processor P_i, and the execution of S_i is independent of any other autonomous services. In the case where multiple services reside on a single processor, a physical processor can serve multiple virtual processors and set appropriate capacity parameters for the virtual processors. Also, complex autonomous services that involve multiple processors for execution can be further partitioned into atomic service units such that each atomic service unit only performs operations on its local processor. As a result, we can establish a simple one-to-one mapping between a processor and an autonomous service for our analysis.

Control-flow and data-flow are both modeled as communication messages, either originated or received at an autonomous service. Autonomous service S_i is invoked by receiving input data of size SI_i, including both control and data. It is executed at a cost of SP_i expressed in terms of number of cycles. An output data of size SO_i is generated as the result of executing S_i, again including both control and data.

A megaservice M is a running instance that specifies a partially ordered sequence of tasks. Without loss of generality, we assume that M runs on processor P_0. In the case where the megaservice M runs on processor P_i ($i \neq 0$), our analysis holds by simply treating the invocation of autonomous service S_i as part of local processing.

A megaservice M is regarded as a transcript of accomplished tasks in our performance analysis. Tasks are classified as either local processing or remote invocations of autonomous services in $\boldsymbol{S}$. The workload of each type of tasks is then evaluated. Local processing takes place on P_0 and the workload is denoted as MP number of cycles. Remote invocations of autonomous services is modeled as a frequency vector $\boldsymbol{F} = \{f_1, \ldots, f_n\}$, where f_i denotes the number of times S_i is invoked during the execution of M.

Given the mathematical model for the computing environment, we proceed to analyze the performance of a megaservice with two specific metrics: the aggregated cost and the response time.

3. AGGREGATED COST

3.1. Aggregated Cost Definition

Before giving a definition of aggregated cost, we need to first define cost function for individual system resource components. In our model, we refer to a cost evaluation function *Cef*, which is formally a mapping defined as the following: Given a megaservice M and a processor network $\boldsymbol{P}$, $Cef(M)$ is the tuple $(\boldsymbol{vp}, \boldsymbol{vm})$, where

- $\boldsymbol{vp} = \{vp_0, \ldots, vp_n\}$, where vp_i is the load in terms of the number of processor cycles consumed by processor P_i.
- $\boldsymbol{vm} = \{vm_{ij} \mid 0 \le i, j \le n\}$, where vm_{ij} is the load due to the message traffic generated from processor P_i and to processor P_j.

Notice that $\boldsymbol{vp}$ and $\boldsymbol{vm}$ represent the processing costs and the communication costs, respectively.

The aggregated cost for a megaservice is defined as the sum of all individual cost components. It measures the amount of system resource consumed by a megaservice. We assume that the processing costs and the messaging costs of a megaservice can be linearly scaled. The weights of the scale given to the processing costs and the messaging costs are given as α and β respectively. Hence, we define the aggregated cost of a megaservice $COST(M)$ over a processor network $\boldsymbol{P}$ as the following:

$$COST(M) = \alpha \times \sum_{i=0}^{n} vp_i + \beta \times \sum_{0 \le i,\, j \le n} vm_{ij}$$

where $(\boldsymbol{vp}, \boldsymbol{vm}) = Cef(M)$ and $\alpha, \beta \ge 0$.

The weights, α and β, can be set to appropriate values to reflect the relative scarcity of the processor resources to the communication resources. In the extreme case where $\alpha = 0$, the system has unlimited processing power and is limited in its network bandwidth. On the other hand, if $\beta = 0$, the system has unlimited networking bandwidth and is limited in its processing power.

According to the definition, the aggregated cost is a linear aggregation of the cost components defined in the cost evaluation

function. Our analysis hence focuses on determining the ***vp*** and the ***vm*** vectors.

3.2. Centralized Data-flow Model

We start with the centralized data-flow model. Processing cost components ***vp*** are computed using Equation 1. Since the megaservice M is the only local process running on P_0, the processing load of processor P_0 equals MP. The processing loads of other processors equal to the execution costs of the autonomous services running on them multiplied by the number of times that the autonomous services are invoked.

$$vp_i = \begin{cases} MP & if \quad i = 0 \\ f_i \times SP_i & if \quad i \neq 0 \end{cases} \tag{1}$$

The calculation of the messaging cost components ***vm*** is show in Equation 2. The only network traffic in the system is caused by the invocation of services. Messages are sent from P_0 to other processors for invocation of autonomous services, and results are returned to P_0 as messages from the processors where the autonomous services execute.

$$vm_{ij} = \begin{cases} f_j \times SI_j & if \quad i = 0, j \neq 0 \\ f_i \times SO_i & if \quad i \neq 0, j = 0 \\ 0 & otherwise \end{cases} \tag{2}$$

The messaging load of the communication link (P_0, P_i) equals the input data load for autonomous service S_i multiplied by the number of times that S_i is invoked. The load of the communication link (P_i, P_0) equals the result data load from autonomous service S_i multiplied by the number of times that S_i is invoked. There is no other network traffic within the system, and hence all other communication links have a message load of 0.

With ***vp*** and ***vm*** determined, we compute the aggregated cost by its definition as shown in Equation 3:

$$\begin{aligned} COST_c(M) &= \alpha \times \sum_{i=0}^{n} vp_i + \beta \times \sum_{0 \le i,j \le n} vm_{ij} \\ &= \alpha \times (MP + \sum_{i=1}^{n} f_i \times SP_i) + \beta \times \sum_{i=1}^{n} f_i \times (SI_i + SO_i) \end{aligned} \tag{3}$$

3.3. Distributed Data-flow Model

The processing cost components ***vp*** are the same in the distributed data-flow model as in the centralized data-flow model. The difference is in the communication cost components. To compute the communication costs, we first model the network traffic pattern within the distributed data-flow model.

A data distribution coefficients vector is defined as $\boldsymbol{\Delta} = \{\delta_{ij} \mid 1 \le i \le n, 0 \le j \le n\}$ that describes the level of distributed data flow among the autonomous services. δ_{ij} is computed as:

$$\delta_{ij} = dd_{ij} / SO_i$$

where dd_{ij} is the size of the output data generated by autonomous service S_i that transmits directly from processor P_i to processor P_j for further processing. The data distribution coefficients have the following property:

$0 \le \delta_{ij} \le 1$, for all $1 \le i \le n$, $0 \le j \le n$.

We would like to point out two special cases regarding the data distribution coefficients. (1) $\delta_{ij} = 0$ for all $1 \le i, j \le n$: The distributed data-flow model converges with the centralized data-flow model, where data-flows only exist between autonomous services S_i and the megaservice; (2) $\delta_{i0} = 0$ for all $1 \le i \le n$: The integration model becomes a fully distributed data-flow model, where all data-flows are established directly between autonomous service, and no data is returned back to the megaservice for processing. The majority of the distributed data-flow integration infrastructures fall between the above special cases.

Given the data distribution coefficient vector, messaging cost components for the distributed data-flow model are computed in Equation 4. There are four types of messaging cost components:

- vm_{0j} refers to the costs on the communication link (P_0, P_j) for invoking autonomous service S_j. The data volume from the megaservice to the autonomous service S_j equals the total size of invocation data SI_j minus the portion contributed by other autonomous services.
- vm_{i0} refers to the costs on the communication link (P_i, P_0) for sending the result data from autonomous service S_i back to the megaservice. The data distribution coefficients are applied to the total output data size generated by S_i.
- vm_{ij} refers to the costs on the communication link (P_i, P_j) for sending the data between autonomous services. For each invocation of autonomous service S_j, data of size $\delta_{ij} \times SO_i$ needs to be sent from P_i to P_j.
- The last messaging cost components refer to the messaging costs from the megaservice to itself, which equals 0.

$$vm_{ij} = \begin{cases} f_j \times (SI_j - \sum_{k=1}^{n} \delta_{kj} \times SO_k) & if \quad i = 0, j \neq 0 \\ f_i \times \delta_{i0} \times SO_i & if \quad i \neq 0, j = 0 \\ f_j \times \delta_{ij} \times SO_i & if \quad i \neq 0, j \neq 0 \\ 0 & if \quad i = 0, j = 0 \end{cases} \tag{4}$$

Combing the cost components, we can derive the aggregated cost for the distributed data-flow model as shown in Equation 5:

$$\begin{aligned} COST_d(M) &= \alpha \times \sum_{i=0}^{n} vp_i + \beta \times \sum_{0 \le i,j \le n} vm_{ij} \\ &= \alpha \times (MP + \sum_{i=1}^{n} f_i \times SP_i) + \beta \times \sum_{i=1}^{n} f_i \times (SI_i + \delta_{i0} \times SO_i) \end{aligned} \tag{5}$$

3.4. Model Comparisons

The following proposition is drawn from the analysis on the aggregate costs of the two data-flow models.

PROPOSITION 1. *The aggregated cost incurred by a megaservice under a distributed data-flow model is no greater than the aggregated cost under a centralized data-flow model.*

Proof. The difference between the two aggregated costs of the centralized and distributed data-flow models can be computed using Equation 3 and Equation 5:

$$COST_c(M) - COST_d(M) = \beta \times \sum_{i=1}^{n} f_i \times SO_i \times (1 - \delta_{i0}) \ge 0$$

We observe that the savings in the aggregated cost of the distributed data-flow model comes from the difference in data passed from autonomous services back to the megaservice, as indicated by the ($1-\delta_{i0}$) factor in the above proof. Based on the

observation, we propose two performance optimization approaches for megaservice execution. The first is to form distributed data-flows by discovering direct input-output data mappings among autonomous service invocations. For instance, in the sample megaservice illustrated in Figure 3, the output of autonomous service *S1* can be mapped to the input of autonomous service *S2*. By sending the output of *S1* to *S2* for processing without going through the megaservice, we can eliminate the potential data traffic between *S1* and the megaservice *M*, reducing δ_{10} to 0.

```
MEGAPROGRAM M {
    b = S1(a)
    c = S2(b)
    d = local-transform(c)
    e = S3(d)
    f  = S4(b)
    result = local-processing(e, f)
}
```

Figure 3: A Sample Megaservice

The second optimization approach utilizes code transfer to reduce data traffic. Code segments are transferred to the most appropriate location for execution to reduce the amount of data communication between autonomous services and the megaservice. Using the example shown in Figure 3, we notice that the output of autonomous service *S2* needs to be processed by *local-transform* routine before passing on to *S3* for further processing. If the *local-transform* routine can be shipped onto and carried out on the processor serving *S2*, the transformed output data for *S2* can be directly sent to *S3*. In order to take advantage of this optimization approach, the integration infrastructure needs to support code-shipping and remote invocation of code segments. Our investigation indicates that active mediation technology [2, 6] can potentially be incorporated into the integration infrastructure to provide the necessary run-time support.

4. RESPONSE TIME

Another aspect of our performance analysis focuses on response time, i.e. the elapsed time between the start and the termination of a megaservice. The response time analysis helps pinpointing bottlenecks of a system and providing a scalable design.

4.1. Overview

We analyze the response times for the two atomic operations performed by a megaservice: processing of computational load *vp* on processor P_i and sending a message of size *vm* from processor P_i to processor P_j. Since the response time for atomic operations equals the workload divided by the system resource bandwidth, the computational operation has a response time of vp/CP_i, and the communication operation has a response time of vm/CM_{ij}.

A megaservice is as a sequence of partially ordered distributed atomic operations. By executing atomic operations in parallel, processors can be better utilized, offering better response time for the megaservice. Since complexities are introduced in the analysis of parallel executions of megaservices, we separate our analysis for the serial and parallel invocation schemes.

The serialized invocation scheme assumes single threaded execution for a megaservice. Autonomous service invocations are carried out in the order they are specified in the megaservice. Given the deterministic nature of the serialized invocation scheme, we compute and compare quantitatively the response times of megaservices. The parallel invocation scheme favors overlapping execution of autonomous services, subject to data dependencies. We will model megaservices in Petri nets and conduct qualitative comparison of the centralized data-flow model against the distributed data-flow model.

Interference of activities within the processor network, such as running instances of other megaservices, can generate conflicts in allocation of system resources, resulting variance in the response time. For our response time analysis, we assume a single-user usage model, which approximates systems under light loads.

4.2. Serialized Invocation Scheme

The response time *T* for megaservice *M* consists of two components: the local processing time *TM* and the autonomous service invocation time *TS*. Under serialized invocation scheme, *TS* equals the sum of individual TS_i, the elapsed time for invoking autonomous service S_i. Each autonomous service consists of three sequential tasks: input task SI_i during which input parameters are prepared, processing task SP_i during which computation is conducted on the input parameters, and output task SO_i during which the results are returned back to the invoker of the autonomous service. Each task is associated with an elapsed time, TSI_i, TSP_i, and TSO_i, respectively.

The relationship among components of the response time is illustrated in Equation 6. The local processing time *TM* remains the same for both the 1C1D and the 1CnD models, whereas the autonomous service invocation time *TS* is expected to be different.

$$\begin{aligned} &T(M) = TM + TS \\ &TM = \frac{MP}{CP_0} \\ &TS = \sum_{i=1}^{n} f_i \times TS_i \\ &TS_i = TSI_i + TSP_i + TSO_i \end{aligned} \tag{6}$$

The response time for *M* under centralized data-flow model is computed in Equation 7. The input elapsed time dedicates for the megaservice *M* sending the input data of size SI_i to the autonomous service. The processing elapsed time dedicates for P_i processing computational load of the autonomous service. The output elapsed time dedicates for S_i sending the output data of size SO_i back to the megaservice *M*. These components are aggregated to get the overall response time T_c for the megaservice.

The components of the total elapsed time are grouped into two, one as processing costs and the other as communication costs. It is clear that all communication costs are incurred for the traffic going through P_0. Hence, in designing centralized data-flow integration infrastructure, it is important to maximize the communication capacity between the processor on which the megaservice is initiated and the other processors on which the autonomous services reside. On the other hand, the communication capacities between autonomous services have no effect on the response time.

$$\begin{aligned} &TSI_i = \frac{SI_i}{CM_{0i}} \\ &TSP_i = \frac{SP_i}{CP_i} \\ &TSO_i = \frac{SO_i}{CM_{i0}} \\ &T_c(M) = \left(\frac{MP}{CP_0} + \sum_{i=1}^{n} f_i \times \frac{SP_i}{CC_i}\right) + \sum_{i=1}^{n} f_i \times \left(\frac{SI_i}{CM_{0i}} + \frac{SO_i}{CM_{i0}}\right) \end{aligned} \tag{7}$$

The response time for *M* under distributed data-flow model is computed in Equation 8. The processing costs of autonomous

services remain the same as in the centralized data-flow model, while the communication costs differ.

$$TSI_i = Max\left\{\frac{SI_i - \sum_{j=1}^{n}\delta_{ji}\times SO_j}{CM_{0i}} \quad , \quad \underset{k=1}{\overset{n}{Max}}\left\{\frac{\delta_{ki}\times SO_k}{CM_{ki}}\right\}\right\}$$

$$TSP_i = \frac{SP_i}{CP_i} \qquad (8)$$

$$TSO_i = \frac{\delta_{i0}\times SO_i}{CM_{i0}}$$

$$T_d(M) = (\frac{MP}{CP_0} + \sum_{i=1}^{n} f_i \times \frac{SP_i}{CC_i}) + \sum_{i=1}^{n} f_i \times (TSI_i + \frac{\delta_{i0}\times SO_i}{CM_{i0}})$$

The input elapsed time for autonomous service S_i refers to the time to prepare the input parameters for the autonomous service. The time equals the maximum length of all the communication processes, during which related autonomous services and the megaservice send their portions of input data to S_i.

The output elapsed time equals the size of output data being sent back from autonomous service S_i to the megaservice M divided by the bandwidth capacity of the communication link. The output elapsed time is guaranteed to at least as short under distributed data-flow model as under the centralized data-flow model, since the values differs by a factor of δ_{i0} under the two integration models.

PROPOSITION 2. The response time incurred by a megaservice under a distributed data-flow model is no greater than the response time under a centralized data-flow model, if the following conditions are met:

- Autonomous services invocations are serialized; and
- $CM_{ki} \geq CM_{0i}$ for all $k \neq 0$ and $i \neq 0$.

Proof. By the definition of data distribution coefficients:

$$0 \leq \delta_{ki}\times SO_k \leq SI_i \quad \text{for all } k \neq 0 \text{ and } i \neq 0$$

Hence, we have:

$$\underset{k=1}{\overset{n}{Max}}\left\{\frac{\delta_{ki}\times SO_k}{CM_{ki}}\right\} \leq \frac{SI_i}{CM_{0i}} \quad \text{and} \quad \frac{SI_i - \sum_{j=1}^{n}\delta_{ji}\times SO_j}{CM_{0i}} \leq \frac{SI_i}{CM_{0i}}$$

Since the response time is a linear aggregation of the elapsed time components, comparing Equation 7 and Equation 8 deduces:

$$T_d(M) \leq T_c(M) \cdot$$

The condition on communication capacities in the above proposition implies a communication backbone among the autonomous services with at least as much bandwidth as between the megaservice and the autonomous services. Many computing networks easily satisfy this condition. For instance, in many intranets that consist of uniformly connected processor networks, all communication links have the same bandwidth. Another example is the client-server network, where autonomous services reside on well-connected server farms and the megaservices are clients accessing the servers from remote sites. The Internet and mobile service networks both can be categorized as this type.

4.3. Parallel Invocation Scheme

We extend the response time analysis to parallel invocation scheme, where causally unrelated autonomous services can be executed in parallel, improving megaservice performance by reducing the overall response time. The megaservice performance is affected by multiple factors, such as the degree of parallelism in a megaservice, the process scheduling algorithms, etc. In order to make relevant comparison on the effects of data-flows between the 1C1D and 1CnD models, we make the assumption that the control-flows remain the same for the integration models used, namely the partial order of autonomous service invocations are identical.

We first model the execution of a megaservice. It has been shown that timed Petri net [12] models can be used as an effective modeling tool for representing synchronization and concurrency [8, 10]. We model the execution of a megaservice as a timed marked graph (TMG), a well-known subclass of Petri nets that allows representations of concurrency and synchronization, but not decision or conflicts. Branches and loops within a megaservice are unfolded during the execution so that a megaservice can be seen as a partially ordered string of execution tasks, each being either an autonomous service or a local processing segment of the megaservice.

Figure 4 illustrates a TMG representation of the megaservice defined in Figure 3. Places, drawn as circles, are used to represent tasks. Transitions, drawn as boxes, are used to represent synchronization points. Places are labeled as one of the following: (1) an input task of an autonomous service (e.g. SI_{1_a}), (2) a processing task of an autonomous service (e.g. SP_{1_a}), (3) an output task of an autonomous service (e.g. SO_{1_a}), or (4) a local processing segment of the megaservice (e.g. $MP_{\_c}$). A unique subscription to distinguish the multiple invocations of an autonomous service is attached to the end of each label.

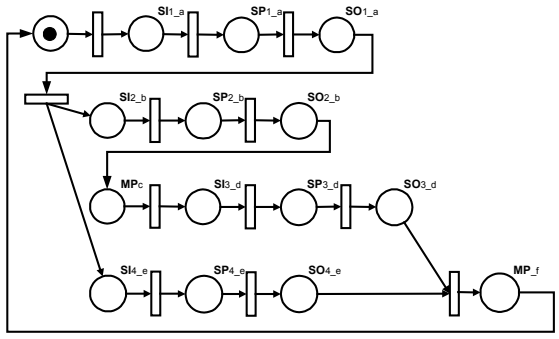

Figure 4: Timed Marked Graph Representation of Megaservice

Each place is assigned a time delay τ_P that equals the elapsed time to perform the task represented by the place P. A single token is placed in the initial place as the starting marking. Such a Petri net model is known as a deterministic timed net, and the response time of the megaservice equals the minimum cycle time of the net.

PROPOSITION 3. The response time incurred by a megaservice under a distributed data-flow model is no greater than the response time under a centralized data-flow model, if the following conditions are met:

- $CM_{ki} \geq CM_{0i}$ for all $k \neq 0$ and $i \neq 0$.

Proof. It was shown in [10] that the minimum cycle time of the TMG equals the maximum of the total delays in all directed circuits. Hence the response time incurred by a megaservice equals the total delay of the longest non-cyclic path in the graph.

Consider two TMGs: TMG^c and TMG^d, representing megaservice executions under the centralized-data flow model and the distributed data-flow model, respectively. By construction, TMG^c and TMG^d have the same structure, i.e. the same set of places, transitions, arcs and initial markings. They differ only in the time delays that are assigned to the places, which are denoted as τ^c_p and τ^d_p for a place p in TMG^c and TMG^d respectively. As shown in Equation 7 and Equation 8, $\tau^c_p \geq \tau^d_p$, if $CM_{ki} \geq CM_{0i}$ for all $k \neq 0$ and $i \neq 0$.

Let's denote the longest non-cyclic path in TMG^d as P. The response time for the distributed control model equals the total delay of the path P in TMG^d, which is $\sum_{p \in P} \tau^d_p$. The response time for the centralized model is no less than the total delay for the path P in TMG^c, which is $\sum_{p \in P} \tau^c_p$.

Since $\sum_{p \in P} \tau^c_p \geq \sum_{p \in P} \tau^d_p$, the proposition holds.

It is clear that Proposition 3 covers Proposition 2, since serialized invocation is a special case of parallel invocation. Therefore, we have proved that the distributed data-flow model has better response time performance than the centralized data-flow model if the system communication capacity condition is met.

5. EXAMPLE SCENARIOS

We study the performance of megaservices in an example engineering service environment. The runtime performances are measured and compared against the results obtained through our analytical models.

Figure 5 illustrates a scenario where three autonomous services are used in collaboration to conduct engineering planning and scheduling. The *ModelRetriever* service fetches a project model from information sources based on the name of the project. The *Scheduler* service generates new schedules for the input project model. The *ChangeManager* service manages the changes introduced by the new schedules. The autonomous services run on distributed servers that are connected via a switch with 10mbps bandwidth on each port.

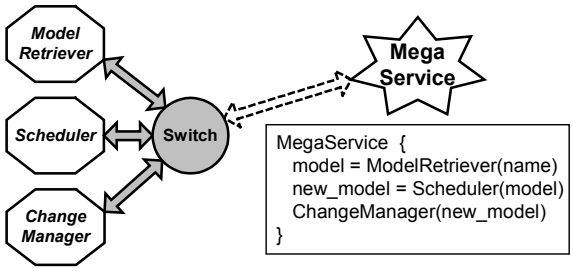

Figure 5: A Megaservice for Engineering Services

The megaservice is composed of three autonomous services. It is designed to retrieve a specific project model using the *ModelRetriever* service, then conduct scheduling on the model using the *Scheduler* service, and finally notify the related parties about the change via the *ChangeManager* service. The level of data-flow distribution can be described by the data distribution coefficients of the megaservice, which can be generated from the megaservice specification. For instance, all the coefficients between megaservice and autonomous services are 0, and the coefficient from *ModelRetriever* to *Scheduler* equals 1.

The megaservice runs on a client machine that is connected to the servers either directly via the switch or via an 802.11 wireless access point. The two network connection settings yield different performance results since the wireless access has much lower communication bandwidth.

We implement the megaservice with two different integration models: (1) SOAP [14] is used as the reference platform for the 1C1D model, where each service invocation is a remote procedure call initiated from the megaservice; and (2) FICAS [7] is used as the reference platform for 1CnD model, where data-flows are formed between autonomous services.

The response times of megaservices are measured with different settings on the size of the project model. Since the computational elapsed times contributed to the autonomous service executions are identical under both integration models, we compare only the communication elapsed time, which is calculated as the megaservice response time minus the sum of processing elapsed times of autonomous services. Figure 6 shows the performances of the megaservice measured with two service access media and two integration models.

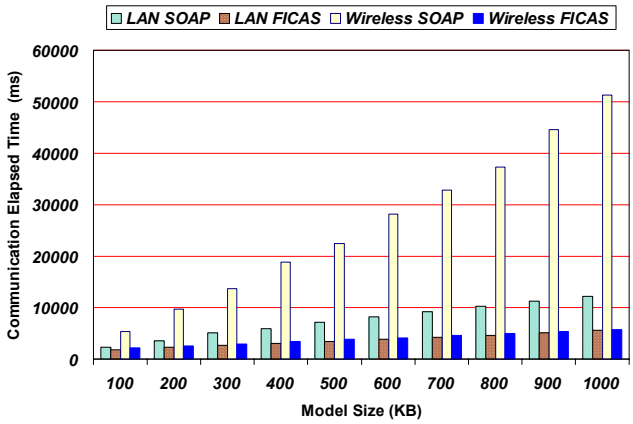

Figure 6: Performance of the Megaservice

We first verify that the real-world behavior of the megaservice is consistent with the prediction of our analytical model. Data points can be used to deduce the system parameters by applying the numbers to the analytical equations. For instance, we use the performance numbers measured for the model size of 1000KB. Applying Equation 7, we calculate the communication capacity on the LAN connection as:

$$\frac{4 \times 1000KB}{12226\text{ms}} \approx 2.6mbps$$

and the communication capacity on the wireless connection as:

$$\frac{4 \times 1000KB}{51286\text{ms}} \approx 0.63mbps$$

Linear regression techniques may also be used on multiple data points when more accurate estimation is desired.

The values of the communication capacities are then applied to Equation 7 and Equation 8 to calculate the communication elapsed times for other model sizes. Our analytical model is a good approximation of the real-world megaservice behavior and the calculated values agree well with the measured values.

The communication capacities between autonomous services are clearly not the same as the network bandwidth between the servers on which the services run. In fact, the effective rate by

which the autonomous service can exchange data is affected by many factors, ranging from the implementation of the integration software, the construction of the server platform, to the layout of the communication network, etc. The example scenario demonstrates how the communication capacities of an integration infrastructure can be estimated with a few measured data points on overall megaservice performance.

Further possible observations can be made that are consistent with our mathematical analysis conducted in earlier sections:

- The response times in the 1CnD model are better than their counterparts in the 1C1D models under all load settings. As we have proven in Proposition 2 and 3, the 1CnD model performs better than the 1C1D model when the system communication capacity condition is met.
- The response time increases linearly with respect to the volume of the data-flows. The response times under the 1C1D model increase at much faster rates than the response times under the 1CnD model. The rate of increase is especially significant in the wireless connection scenario, where data communications between client machines and servers become a bottleneck in the 1C1D model. On the other hand, we observe small increase in response time in the 1CnD model with large model sizes. The 1CnD model alleviates the bottleneck by distributing network traffic within the server farm.

6. CONCLUSIONS

With the advances in computer system and software component technology, there will be increasing interests in infrastructures that facilitate service composition. Service integration infrastructures are classified into four complementary models based on how controls and data are exchanged. We have focused our study on the two models with centralized control-flow structure, partly due to our belief that the ease of use makes the models easier to be adopted for service composition.

Performance analysis on service integration models with centralized and distributed data-flows is conducted in terms of the aggregated cost and the response time metrics. We have shown that the distributed data-flow model always has better performance in aggregated cost. Also, the distributed data-flow model has better performance in response time with uniformly connected processor networks and client-server networks, where the autonomous services have a communication backbone with at least as much bandwidth as the communication channels from the megaservice to the autonomous services.

During the course of our analysis, we have identified a few critical system parameters that correlate with the performance of megaservice executions. Techniques to improve performance of service integration infrastructures are introduced. First, performance can be improved by distributing data-flows among autonomous services, hence reducing the amount of data-flows between megaservices and autonomous services. Secondly, we motivated the use of dynamic code-shipping technique to conduct remote processing of data. The technique can establish more distributed data-flows and reduce the communication traffic between megaservices and autonomous services even further. Finally, we identified system bottlenecks via the response time analysis. The findings will help guide building appropriate system architectures for composing autonomous services.

7. ACKNOWLEDGEMENT

This research is partially sponsored by the National Institute of Standards and Technology and Center for Integrated Facility Engineering at Stanford University.

8. REFERENCES

[1] B. Boehm and B. Scherlis, "Megaprogramming", Proceedings of DARPA Software Technology Conference, Los Angeles, April 1992, pp. 68-82.

[2] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing", *IEEE Personal Communications*, vol. 2(5), October 1995, pp. 34-49.

[3] J. B. Dennis and G. R. Gao, "An Efficient Pipelined Dataflow Processor Architecture", Proceedings of Supercomputing '88, IEEE Computer Society Press, November 1988, pp. 368-373.

[4] J. B. Dennis and D. P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor", Proceedings of 2nd Annual Symposium on Computer Architecture, New York, 1975.

[5] J.-L. Gaudiot and L. Bic (eds.), "Advanced Topics in Data-Flow Computing", Prentice-Hall, 1991.

[6] D. Liu, K. Law, and G. Wiederhold, "CHAOS: An Active Security Mediation System", Proceedings of International Conference on Advanced Information Systems Engineering, LNCS, vol.1789, B. Wangler and L. Bergman (eds.), Springer-Verlag, 2000, pp. 232-246.

[7] D. Liu, K. Law, and G. Wiederhold, "FICAS: A Distributed Data-Flow Service Composition Infrastructure", Stanford University, Unpublished Report, 2002, http://mediator.stanford.edu/papers/FICAS.pdf.

[8] J. Magott, "Performance Evalucation of Concurrent Systems Using Petri Nets", *Information Processing Letter*, vol. 18(1), 1984, pp. 7-13.

[9] M. D. McIlroy, "Mass Produced Software Components", *Software Engineering, NATO Science Committee*, January 1969, pp. 138-150.

[10] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, vol. 77(4), April 1989, pp. 541-580.

[11] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *P. Freeman and A. I. Wasserman*, Tutorial on Software Design Techniques, IEEE Computer Society Press, 1983.

[12] C. Petri, "Kommunikation mit Automaten", University of Bonn, Ph.D. dissertation, 1962.

[13] A. H. Veen, "Data Flow Machine Architecture", *ACM Computing Surveys*, December 1986.

[14] W3C, "Simple Object Access Protocol (SOAP)", 2000, http://www.w3.org/TR/SOAP.

[15] G. Wiederhold, D. Beringer, N. Sample, and L. Melloul, "Composition of Multi-site Services", Proceedings of IDPT'99, Kusadasi, Turkey, June 1999.

[16] G. Wiederhold and R. Jiang, "Information Systems That Really Support Decision-Making", *Journal of Intelligent Information Systems*, vol. 14, March 2000, pp. 85-94.

[17] G. Wiederhold, P. Wegner, and S. Ceri, "Towards Megaprogramming", *Comm. ACM*, vol. 35(11), Nov 1992, pp. 89-99.