# Performance Evaluation of Choreographies and Orchestrations with a New Simulator for Service Compositions

Felipe Pontes Guimaraes
Department of Computer Science
University of Sao Paulo
Sao Paulo, Brasil
E-mail: felipepg@ime.usp.br

Eduardo Hideo Kuroda
Department of Computer Science
University of Sao Paulo
Sao Paulo, Brasil
E-mail: eduardok@ime.usp.br

Daniel Macedo Batista
Department of Computer Science
University of Sao Paulo
Sao Paulo, Brasil
E-mail: batista@ime.usp.br

***Abstract*—Service-oriented architecture was designed to support the development of massively distributed applications that communicate over the Internet. Different services may be composed using service compositions schemes, mainly orchestrations and choreographies. However, to evaluate how the characteristics of the environment affect the QoS requirements of the compositions is a difficult task. To be able to do so, we developed a SOA simulator, estimated and compared the performance for two real-world scenarios developed using orchestrated and choreographed implementations. Proper choosing of the service composition may incur in significant performance enhancement. In our experiments, the average response time using choreographies was up to 50% lower than its orchestrated counterpart. Besides, for the same available bandwidth, choreographies presented response time lower than orchestrations.**

## I. Introduction

The internet is no longer a very slow and cumbersome environment but instead, is getting ever greater focus and usability. Now users are using many different classes of devices to connect and interact with the network. These devices may vary from dedicated servers, personal computers and notebooks to cell phones, probes, sensors and even low cost music players. This phenomena is usually referred to as "the Internet of Things" [1]. In these times of ever-growing available bandwidth, network traffic and heterogeneous connected devices, application design also had to evolve. This evolution in design came primarily with Service-Oriented Architecture (SOA) and the Future Service-Oriented Internet.

SOA was designed to support the development of rapid, low-cost, interoperable and massively distributed applications [2]. It allows that any piece of code and any application component deployed on a system be transformed into a network-available service.

In SOA, different components may work toward a goal in different manners. Sometimes a service coordinates the execution of other services, thus creating what is referred to as an *orchestration*. In other cases, there is no central point of control and each service knows what it is supposed to do, and what other services it must interact with. This kind of interaction is called a *choreography*.

Due to the increasing number of devices joining the internet, a centralized approach like an orchestration may not be sufficiently scalable in terms of network bandwidth to deal with the ever escalating number of devices and services that may be available. Within such a scenario, a decentralized approach, like choreographies, may turn out to be more capable of dealing with such high complexity.

A choreography describes many roles which must be played by the available services. Note that there may be many equivalent services implementing the same role. As an example, a particular service provider may use cloud and elastic computing to dynamically dimension its infrastructure according to the current load, adding or removing instances on-the-fly. This brings many possibilities: one could replace a faulty service in a choreography with another instance of the same service, balance the load among available service providers, choose a service based on the choreography's QoS constraints, etc.

In spite of the apparent choreography advantage due to the distribution of load among the peers, to the best of our knowledge there is no scientific work comparing it against orchestrations in terms of network metrics.

In this paper, we present a comparison between service orchestration and choreography. Both scenarios were simulated on a SimGrid's [3] enhanced version implemented by the authors, which also represents a contribution of the paper. The results show the effectiveness of using a decentralized and service oriented approach to achieve better service response times. For example, in our experiments, to match the performance achieved by the decentralized approach in a scenario with 30Mbps of available bandwidth, the centralized approach needed 770Mbps – 25 times more. For the same bandwidth, the decentralization incurred from 54.98% up to 72.42% faster response time.

This paper is organized as follows: In Section II we present a bibliographic review with related issues and an overview of the state-of-the-art. In Section III we summarize the motivation of the work. In Section IV we present the implemented SOA simulator and describe the simulated scenarios to compare orchestrations and choreographies and in Section V, the results

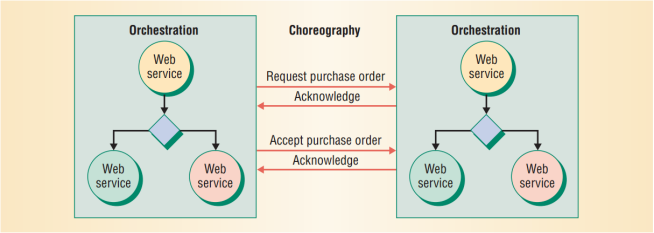



Fig. 1. Choreography and orchestration [5]

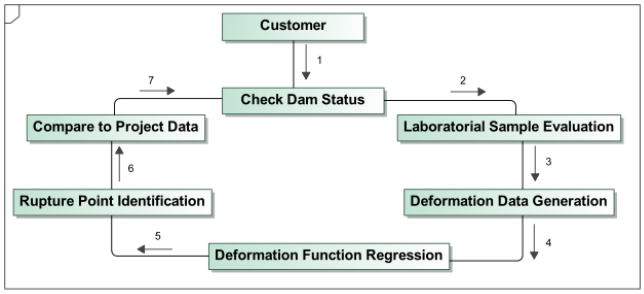



Fig. 2. A choreography for river dam maintenance and disaster avoidance

of the experiments are discussed. In Section VI we present conclusions and future work.

## II. Related Work

### A. Service-Oriented Architecture

The service-oriented architecture supports the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications [2]. It is based upon the architectural approach of loosely coupled services [4].

The visionary promise of SOA is that it will be possible to easily assemble application components into a loosely coupled network of services that can create dynamic business process and agile applications that span organizations and computing platforms. This vision is supported by SOA's basic architecture, where a client may search the Universal Description, Discovery and Integration (UDDI) for a service implementing the required Web Service Description Language (WSDL), bind itself to the provider and make the requests.

Web services usually use the Internet and open-standards, *eg.* SOAP and REST protocols, for transmitting data, and the WSDL for defining services [2].

### B. Service Compositions

In the service-oriented computing paradigm, two major concepts, illustrated in Figure 1, are *orchestration* and *choreography* of services. These concepts have a certain overlap, but the distinction is made from the control point-of-view [5]. *Orchestrations* imply that one party has control over the overall execution. On the other hand, in a *choreography*, the control is not centralized. A choreography is more collaborative and allows each party to describe its part in the interaction [5]. The part that each party take in the choreography is commonly known as the *role* it is playing in the enactment of the choreography.

A choreography example can be viewed in Figure 2. In this choreography, an engineer responsible for a dam decides to monitor it for rupture points in an automated way. A samples collection protocol is started (2). This protocol captures and analyses approximately 6 samples. These samples are sent to a laboratorial evaluation (3) and over 300 measurements for each of those samples are taken. Such measurements are then forwarded to a mathematical regression analysis system (4) that returns a curve describing the data. This function is sent over to a rupture point detection system (5) that figures out the potential rupture points. Finally, such points are compared to the project design and specifications (6) and any needed intervention (repairs, flow control or even immediate evacuation of the area) are listed (7) and returned to the engineer.

### C. Distributed, SOA and Cloud Simulators

Many simulators for distributed environments were proposed, *eg.*, gridsim framework [6], Pi4SOA [7], and SimGrid framework [3].

The gridsim framework [6] is a distributed environment simulation engine based upon on events. It implements entities to emulate users. Users' requests are scheduled through a broker that allocates them into the simulated resources.

Pi4SOA [7] presents a policy-based infrastructure to dynamically verify and control the collaboration process in SOA. It is also used as a starting point to develop an event driven policy enforcement in [8].

The SimGrid framework [3] is a simulation-based framework for evaluating cluster, grid and P2P mechanisms. SimGrid uses tasks to perform the simulation. Such tasks have an intrinsic cost to be transmitted over the network and an execution cost. Resources are described through an XML file in which it is possible to list available resources and their characteristics such as computing power, as well as available links and routes to other resources. In another XML file the deployment of the simulated entities is described, *ie*, where each simulated entity, also referred to as a `Process`, is deployed. Since the SimGrid allows the simulation of distributed environments, we used it as the base to implement our simulator.

## III. Motivation

Many efforts and standards have been emerging to deal with choreographies. Although the premise that choreographies have greater possibility to scale over the number of requests and resources, little has been done to effectively and numerically demonstrate such a premise [9].

A way to compare these two approaches is to simulate them and evaluate all the metrics related to scalability. As presented in Section II-C, there is no simulator available to realize such experiments. Because of that, this paper develops, presents and uses a simulated environment to compare both approaches.

## IV. SOA Simulations

To be able to perform the SOA simulations to compare orchestrations and choreographies performance, we developed a new simulator. It is composed by a set of entities implemented on top of SimGrid's Java bindings.

The services were modelled as a set of working threads that receive a task sent over the network, execute it and then send another task over the network to act as a web service response. The available methods and computational effort needed to execute them, the amount of worker threads, medium size of the given responses, and service name are configurable through a deployment XML file.

Orchestrators have the full orchestration coded into them, and perform the service invocations accordingly. On the other hand, an entity called Coordinate/Delegate or simply Coordel was introduced for the choreography implementation. The Coordel is responsible for receiving a request, invoking the actual web service and forwarding the response to the next Coordel.

The developed simulator was used to compare the performance of a choreography and an orchestration. Available benchmarks, for instance Wsben [10], do not provide a workflow testbed. Thus two distinct distributed SOA scenarios envisioned by the authors were proposed as an initial testbed.

In the first scenario, a client wishes to purchase some goods from a supermarket. The client provides a list of products and supermarkets from which to buy them. Each product is then purchased from the provided supermarket. The payment process is then handled by the client's bank and, finally, a shipper service is invoked to deliver the items.

The second scenario regards the river dam control and disaster avoidance described in Section II-B.

*1) Deployment:* Both scenarios were deployed in the simulator with an orchestrated implementation and a choreographed implementation. In all deployments, the previous existence of the Web Services is assumed. Also, it is assumed that no intervention on the actual web service should be needed. Moreover, it is assumed that such an intervention should be deemed unfeasible since in a real world scenario this would probably be the case.

*a) Orchestration:* The orchestrated scenarios were implemented by choosing an entity to act as the service orchestrator and handle all successive invocations of services.

For the supermarket scenario, as presented in Figure 3, a `Costumer` asks the `Orchestrator` for a product purchase (1 – `makePurchase`) and the orchestrator invokes the cheapest supermarket service for each product (1.1 – `purchase`), makes the purchase on it by interacting with the `Bank` WS to handle payment issues (1.2 – `requestPayment`) and with the `Shipper` WS to properly deliver the purchased goods to the purchasing Customer (1.3 – `setDelivery`).

The river dam control and disaster avoidance was implemented in a similar way. Since this is a simpler scenario, the `Dam Status Checker` serves as the orchestrator, invokes the services in the proper order and replies to the Costumer, as portrayed in Figure 4.

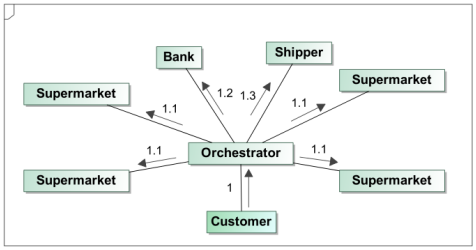



Fig. 3. Orchestrated implementation for the supermarket scenario

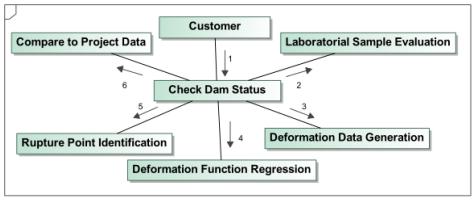



Fig. 4. Orchestrated implementation for the river dam scenario

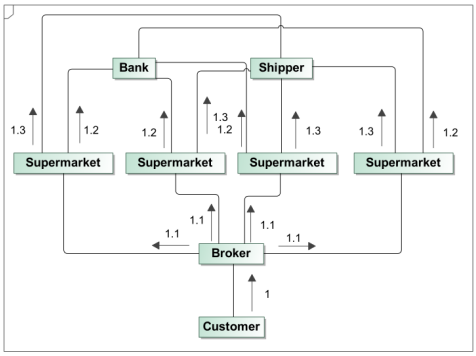



Fig. 5. Choreographed implementation for the supermarket scenario

*b) Choreography:* The choreographed implementation for the supermarket scenario is somewhat similar to the orchestrated implementation but has a key difference. Now there is no central point of control. The `Orchestrator` entity is removed and a less empowered entity, namely `Broker`, is added.

For each service in the choreography, an specific Coordel was tailored.

*2) Simulated Environment:* In the supermarket choreographed scenario, the `Broker` entity does not deal with the shipment and payment issues. As it can be seen from Figure 5, the `Broker` only interacts with the `Supermarket` web services to request the purchases (1.1). The supermarket services themselves interact with the `Bank` and `Shipper` services to handle payment (1.2) and deliver (1.3) the purchased goods.

For the river dam scenario (Figure 2), each `Coordel` is aware of its role in the choreography. Therefore, each `Coordel` receives the input, invokes the corresponding service and forwards the output to the next `Coordel`. This is repeated until the `Compare Data Against Dam Specs` forwards its output to the initial `Coordel` and the choreog-


(a) Request quantity

(b) Arrival rate

Fig. 6. Supermarket scenario - average response time as a function of number of requests (a) and arrival rate (b)

raphy is done.

Aiming at having the simulated environment as close as possible to a real one, the communication and computational resources were defined similar to the ones provided by HP's OpenCirrus grid computing platform [11].

To achieve such a goal, actual web services were deployed in OpenCirrus, then measurements such as response time, bandwidth, latency and processing power in flop/s were taken. Such measures were then fed to our simulator via an XML file.

From these measures, approximate computing time for each web service as well as input and output sizes were also estimated. These estimates were also provided to SimGrid's simulation engine through arguments in another XML file.

In total, fifteen computing nodes were simulated. Each having a Quad-core Intel(R) Xeon(R) CPU with 3.0GHz each. The available bandwidth between the hosts was 943 Mbps.

## V. Results

In the simulations, we distributed each `service`, `orchestrator` or `Coordel` on an exclusive simulated resource.

Figure 6a shows the relationship between the amount of parallel requests and the average response time provided by each implementation. It can be seen that, even though orchestrated and choreographed implementations for the supermarket scenario were very similar, choreography did offer a gain of approximately 4,6% over the orchestrated version with 1000 requests.

In Figure 6b, the relationship between the arrival rate of requests (in requests/second) and the average response time is presented. The behavior for both orchestrated and choreographed implementations are similar but the orchestrated version is lagging behind.

The minor difference perceived on the data is due to the fact that both orchestrated and choreographed versions are very similar in nature and that network traffic is small (between 250 and 1375 bytes) and flows through a nearly gigabit network interface.

In contrast with the supermarket scenario, the river dam control and disaster avoidance scenario is a more burdensome scenario where the network traffic is significantly higher, with some tasks outputting files as big as 500 MB. Also, some of the tasks of this workflow are more computationally-bounded and may take up to 5 minutes.

As presented in Figure 7, in this more resource consuming scenario, the difference between both implementations becomes really significant. As presented in Figure 7c, the larger the load, the greater the difference between choreographed and orchestrated implementations becomes. With a single request, the choreographed version was 1.26% faster than the orchestrated alternative. However, this difference increased rapidly, reaching up to 55.14% with 1000 parallel requests.

However, the arrival rate of requests had little impact on the observed performance. This was attributed to the fact that this is a very long lasting workflow (see Figure 7b). This is because a very small amount of requests already makes the system saturated, so any arrival rate greater than 3 requests per second will only incur in requests being queued.

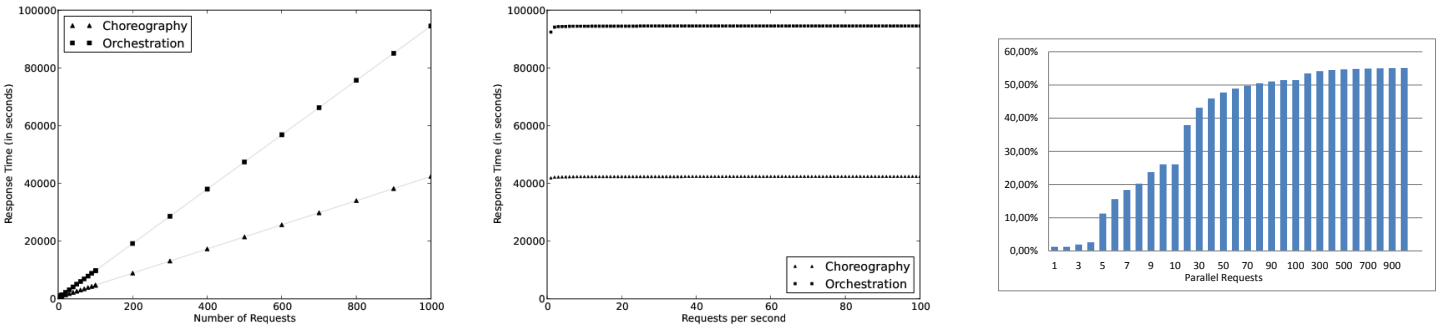
Finally, in Figure 8 the relationship between the available bandwidth and the response time for orchestration and choreography is presented. It shows that for the river dam scenario, the orchestrated version's response time decreases significantly to increases in bandwidth from very low bandwidths up to 200 Mbps. At this point, the overall performance reaches an apparent lower bound on the performance.

On the other hand, the choreography responds even more rapidly to increases in network bandwidth, achieving the same performance as the orchestration in much lower bandwidths. Moreover, the choreographed version reaches an even smaller lower bound much earlier than the orchestrated version.

## VI. Conclusion and Future Work

In this paper, we introduced a simulator for composite services environments. Aided by this tool, we simulated two different service composition techniques, namely orchestration and choreography. Two different real-world scenarios were implemented using both approaches and the simulation results were compared.

In all experiments, choreographies revealed to be a better choice with regard to performance. This advantage was present




(a) Request quantity (b) Arrival rate (c) Gain in performance due to choreography

Fig. 7. River dam scenario - average response time as a function of number of requests (a) and arrival rate (b) and the proportional gain (c)

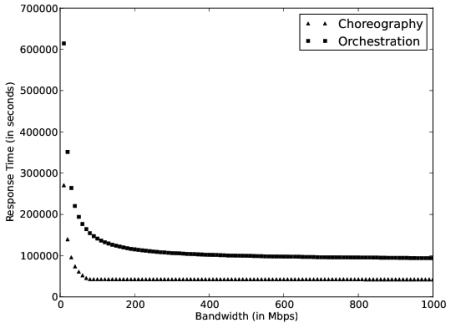



Fig. 8. River dam scenario - Bandwidth impact on the response time

even in an inherently centralized context like the supermarket scenario and greatly increases the overall performance in more processing and network consuming scenarios like the river dam control and disaster avoidance.

As a future work we will derive an analytical model to orchestrations and choreographies.

## ACKNOWLEDGMENT

This work is supported by CAPES, CNPq, HP Brasil under technical cooperation agreement number HP-045/12 (Baile Project - http://ccsl.ime.usp.br/baile/) and the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (Project CHOReOS - Large Scale Choreographies for the Future Internet - http://www.choreos.eu).

## REFERENCES

[1] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things." *Scientific American*, vol. 291, no. 4, pp. 76–81, 2004.

[2] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[3] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: a generic framework for large-scale distributed experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, mar 2008.

[4] S. Ross-Talbot, "Orchestration and choreography: Standards, tools and technologies for distributed workflows," in *NETTAB Workshop-Workflows management: new abilities for the biological information overflow, Naples, Italy*. Citeseer, 2005.

[5] C. Peltz, "Web services orchestration and choreography," *Computer*, pp. 46–52, 2003.

[6] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[7] X. Zhou, W. Tsai, X. Wei, Y. Chen, and B. Xiao, "Pi4soa: A policy infrastructure for verification and control of service collaboration," in *e-Business Engineering, 2006. ICEBE'06. IEEE International Conference on*. IEEE, 2006, pp. 307–314.

[8] W. Tsai, X. Zhou, and Y. Chen, "Soa simulation and verification by event-driven policy enforcement," in *Simulation Symposium, 2008. ANSS 2008. 41st Annual*. IEEE, 2008, pp. 165–172.

[9] A. Barker, C. D. Walton, and D. Robertson, "Choreographing Web Services," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 152–166, Apr. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4815204

[10] S. Oh, H. Kil, D. Lee, and S. Kumara, "Wsben: A web services discovery and composition benchmark," in *Web Services, 2006. ICWS'06. International Conference on*. IEEE, 2006, pp. 239–248.

[11] R. Campbell, I. Gupta, M. Heath, S. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Lee, M. Lyons *et al.*, "Open cirrustm cloud computing testbed: federated data centers for open source systems and services research," in *Proceedings of the 2009 conference on Hot topics in cloud computing*. USENIX Association, 2009, pp. 1–1.