WS-Gossip: Middleware for Scalable Service Coordination

Filipe Campos
Qimonda Portugal S.A.
(Trainee/Internship, 2008)
fcampos@di.uminho.pt

José Pereira
Universidade do Minho
jop@di.uminho.pt

ABSTRACT

The evolution and growing adoption of service-oriented computing increases the demand for applications involving the coordination of very large numbers of services. The goal of WS–Gossip is to leverage gossiping in service-oriented computing as a high level structuring paradigm, thus inherently achieving scalability and resilience when coordinating large numbers of services.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; D.2.11 [**Software Architectures**]: Patterns

General Terms

Design, Performance, Reliability

Keywords

Web Services, Gossip

1. MOTIVATION

The evolution, as well as the growing adoption, of service-oriented computing represents an increase in the demand for applications involving very large numbers of services which can be coordinated through some sort of information exchange.

A stock market scenario, where information flows among several nodes of the system, becomes increasingly interesting from a service-oriented point of view, as several markets and trading systems become increasingly interconnected and interoperable, having already motivated multiple research efforts [7, 5, 4].

These systems have very stringent resilience and scalability requirements, that are hard to achieve even with existing monolithic implementations [8]. However, these requirements can be achieved by using gossip-based protocols, like in the case of stable high throughput [2].

Middleware '08 Companion, December 1-5, 2008, Leuven, Belgium

Current state-of-the-art is that stable high throughput can be achieved by using gossip-based, or epidemic, protocols [2]. Such protocols are also highly resilient to network and process faults, while scaling to large number of participants and high message throughput. Gossip protocols are, for instance, a key technology within Amazon.com Web Services implementation infrastructure [9].

The goal of WS–Gossip is to leverage gossiping in service-oriented computing as a high level structuring paradigm, thus inherently achieving scalability and resilience when coordinating large numbers of services. In detail, we aim at using gossip regardless of the system being architected according to existing event dissemination and notification standards, and with minimal to none application code changes.

2. GOSSIP-BASED PROTOCOLS

In a gossip or epidemic protocol, all the processes that make part of a system are potential disseminators of messages. Briefly, every process chooses randomly a subset of the remaining processes to which the message is then forwarded. Each of these processes behaves exactly in the same way when it receives a message. There is no *reactive* mechanism to deal with failures. This also mimics how epidemics spread in populations, hence the name epidemic protocols. Key parameters are:

Fanout (f) Number of targets that are locally selected by each process for gossiping.

Rounds (r) Maximum number of times a message is forwarded before being ignored.

The reliability of these algorithms is based on a *pro-active* mechanism where redundancy and randomization are used to avoid potential process and network link failures. It has also been shown that parameters f and r can be configured [6] such that any desired average number of receivers successfully get the message. Better yet, parameters can be set such that the message is atomically delivered to receivers with high probability.

3. A GOSSIP SERVICE

An example of the WS–Gossip framework is the WS–PushGossip[3], for push-based information dissemination. It works just like *push gossip*, where a node that becomes aware of some new information conveys it to a subset of selected nodes. It is built on the standard WS–Coordination[1] in order to provide gossip-based communication seamlessly

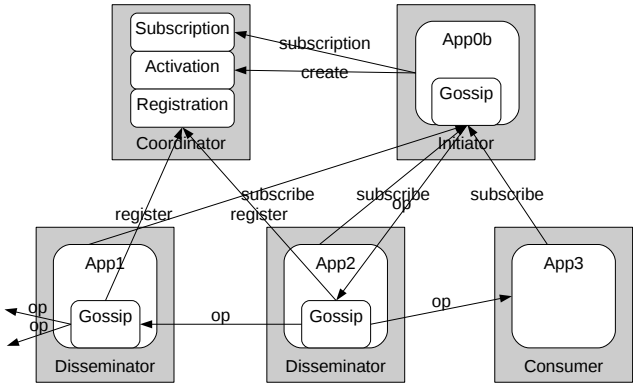

Figure 1: Dissemination using the gossip service.

to any regular service that wishes to disseminate any invocation or result. Figure 1 presents an overview of its architecture. There are four different roles:

Initiator Initiates the dissemination of some data item. This role requires that the application code (App0b) is changed to use the gossip service and that a compliant middleware stack is used.

Disseminator A node that receives a message, sends it to the peers in the list obtained from the Membership service. Although the application code is oblivious to the gossip service, a compliant middleware stack is required.

Consumer A node that receives a message, consumes it. This node is completely unchanged and unaffected by the introduction of gossip.

Coordinator Besides the Activation and Registration services from WS–Coordination, these nodes manage the subscription list.

The main impact of adopting WS–PushGossip is changing the code of the Initiator application to delegate subscription management and to issue a single notification, after having activated a gossip interaction with the Activation service. For a Consumer there is no impact, whereas for a Disseminator it will require configuring an additional handler, the gossip layer, in the middleware stack, which intercepts the outgoing message and re-routes it to selected destinations, i.e. App2 in Figure 1.

Upon arrival to App2, the message is again intercepted by the gossip layer in the middleware stack. If this is an unknown gossip interaction, it registers itself with the Registration service, thus obtaining gossip targets to which it will forward the message. In the case of Figure 1, App1 and App3.

Assuming a single instance of the Coordinator for simplicity, it knows the entire list of subscribers, as well as those that are participating in gossiping. It is thus capable of providing adequate parameter configurations and peers for each gossip round.

Notice that a distributed Coordinator is supported by WS–Coordination and thus also by WS–Gossip, as the list of subscribers can be maintained in a distributed fashion as proposed by WS–Membership [10].

4. CONCLUSION

It is well known in the distributed systems community that stable high throughput information dissemination in large scale heterogeneous systems is a hard problem. Current state-of-the-art points towards gossip-based protocols as the best option regarding scalability and resilience.

In order to avoid these issues, our contribution relies on leveraging gossip-based protocols as a high-level structuring paradigm in a way that they can be regarded as a coordination problem. We propose WS–Gossip, a gossip-based service coordination framework, encompassing different gossip styles and suitable for multiple application scenarios fully integrated in the Web Service ecosystem.

5. REFERENCES

[1] WS-Coordination 1.1 Specification. `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os.pdf`, 12 July 2007.

[2] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[3] F. Campos and J. Pereira. Gossip-based service coordination for scalability and resilience. *MW4SOC 2008*, December 1 2008.

[4] A. Erradi, P. Maheshwari, and V. Tosic. WS-Policy based Monitoring of Composite Web Services. *Web Services, 2007. ECOWS '07. Fifth European Conference on*, pages 99–108, Nov. 2007.

[5] A. Erradi, V. Tosic, and P. Maheshwari. MASC - .NET-Based Middleware for Adaptive Composite Web Services. *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 727–734, July 2007.

[6] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, May 2004.

[7] G. S. Niblett, P. Events and service-oriented architecture: The OASIS Web Services Notification specifications. *IBM Systems Journal*, 44(4):869–886, 25 October 2005.

[8] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange trading system. *Fault-Tolerant Computing, 1997. FTCS-27. Digest of Papers., Twenty-Seventh Annual International Symposium on*, pages 309–313, Jun 1997.

[9] W. Vogels. All Things Distributed. `http://www.allthingsdistributed.com/`.

[10] W. Vogels and C. Re. WS-Membership - Failure Management in a Web-Services World. 2003.