Maintaining Replicated Recovery Log for RESTful Services

Anna Kobusińska, Institute of Computing Science, Poznań University of Technology, Poznań, Poland
Dariusz Wawrzyniak, Institute of Computing Science, Poznań University of Technology, Poznań, Poland

ABSTRACT
Along with development of information systems, their requirements in terms of fault-tolerance increase and become more stringent. A possible approach to deal with this issue is rollback-recovery that consists in loading by the recovering node its most recent checkpoint, or retransmitting requests saved in its message log, to reach a consistent pre-failure state. Both checkpoint and message log are commonly said to be saved in a persistent storage able to survive any failure. In this paper the authors propose the implementation of a stable storage by means of replication of the log containing recovery information. The proposed solution is especially tailored for the service oriented systems implemented accordingly to REST paradigm. Thus, they utilize RESTful semantics in order to reduce the size of replicated recovery log and thereby increasing the efficiency of the proposed recovery log replication protocol.

Keywords: Consistency, Message Log, Replication, REST, Rollback-Recovery, SOA

1. INTRODUCTION
A common problem of remote services in a distributed system is their reliability. The problem results from server crashes as well as communication breakdowns. It becomes more troublesome if there are functional dependencies between services while servers and communication channels fail independently. The request/reply communication scheme, commonly used in access to remote resources in distributed systems (Coulouris, Dollimore, Kindberg, & Blair, 2011; Hwang, Dongarra, & Fox, 2011; Liskov, 1979), in such circumstances gives no reply to the client issuing the access. Missing reply provides no information about the actual reason of the problem, hence reliability is achieved by request re-transmission. However, this may occur not sufficient in the case of stateful services. Two additional complementary techniques are required for this purpose: checkpointing and message logging (Elmootazbellah, Elnozahy, Lorenzo, Wang, & Johnson, 2002). The former consists in saving the current state of a component so as to survive the crash and restore the saved state. The latter allows the recovery of the most recent state after

DOI: 10.4018/IJGHPC.2015070102
rollback to the checkpoint, by repeating the actions that have appeared after the checkpoint was taken. Generally, assuming piece-wise deterministic system, all nondeterministic events must be recorded in a log, and subsequent actions must be replayed to recover a consistent state. Similarly to the checkpoint, the log must survive the crash.

While checkpointing is widely discussed in the literature (Elmootazbellah et al., 2002; Limam & Belalem, 2014; Shahzad et al., 2013; Schelter, Ewen, Tzoumas, & Markl, 2013; Cao & Singhal, 1998), the robustness of log is usually understated or evaded by the assumption of persistent storage for logged messages. Persistent storage supports reliability, however, because of significant mean time to recovery (MTTR), its crash results in log (thereby service) unavailability for a substantial period of time. This way, another point vulnerable to crash is introduced, which can reduce system availability. Log replication can be considered as an alternative or supplementary approach. Replication of log makes the logging mechanism continuously ready, even when some replicas crash. The point at issue is how to exploit the replicas in order to increase log dependability, still avoiding significant overhead. The idea of log replication is rather straightforward — to store each message in an independent log replica. However, in the context of autonomous remote services, especially in SOA-based systems, the way to achieve it is not so obvious. To avoid compromising the autonomy of services, the log should not be a part of the provider itself. Hence, the log replicas should be maintained by some independent services. This raises further questions about dissemination of messages, synchronization of log replicas, handling of replica failures. To answer the above questions, we develop a replication protocol for recovery log, which aims at increasing the reliability of a log of recovery information. In the proposed protocol, in order to intercept messages, the logging mechanism is implemented as a proxy. It represents the genuine service to the consumers and supplies the provider with missing messages in the case of recovery necessity. The solution proposed so far was presented in (Kobusińska & Wawrzyniak, in press) and is general — it does not take into account the characteristic properties neither of SOAP-based web services (Box et al., 2000), nor RESTful (Representational State Transfer) web services (Fielding, 2000; Richardson & Ruby, 2007). In turn, in this paper we focus on RESTful web services, and utilize the RESTful semantics of operations to precisely choose requests, which can be omitted during the service recovery without violating the REST constraints. Consequently, such requests need not to be replicated, and thus the replication protocol efficiency may be increased.

The remainder of this paper is organized as follows. First, the system model and basic definitions are presented. The following sections discuss how to update the log contents accordingly to RESTful constraints and describe the recovery log replication which takes into account the proposed update procedure. Next, the related work is discussed. Finally, the paper is concluded and the future directions of our work are presented.

2. SYSTEM MODEL

The system model assumes two communicating sites: service clients and service providers (simply servers). Service construction follows the Representational State Transfer (REST) architectural style (Fielding, 2000; Richardson & Ruby, 2007). Services of this kind are exposed as a set of resources, which are identified by Uniform Resource Identifiers (URIs). We assume that resources form a tree structure — the dependencies between them are represented as a directed acyclic graph, in which each node has exactly one input edge, and any number of output edges. Each resource identified by \( uri \) can have subordinate resources, denoted by \( S(\text{uri}) \). The subordination path from resource identified by \( uri_i \) to resource identified by \( uri_j \) is a finite
Related Content

Modeling and Simulation of IEEE 802.11g using OMNeT++
www.igi-global.com/chapter/modeling-simulation-ieee-802-11g/38270?camid=4v1a

Grid-Based Nuclear Physics Applications
www.igi-global.com/chapter/grid-based-nuclear-physics-applications/20521?camid=4v1a
Adaptive Self-Organization in Distributed Tree Topologies
www.igi-global.com/article/adaptive-self-organization-in-distributed-tree-topologies/117168?camid=4v1a

Methodology for Information Management and Data Assessment in Cloud Environments
www.igi-global.com/article/methodology-for-information-management-and-data-assessment-in-cloud-environments/127374?camid=4v1a