A Path-Diversity Protocol for the Invocation of Distributed Transactions over the Web

Paolo Romano and Francesco Quaglia
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”

Abstract

The notion of path-diversity has been widely used in the context of Content Delivery Networks (CDNs) to improve system responsiveness by leveraging inherent features of multi-hop networks to reduce the likelihood to incur link congestions or failures. In this work we address the issue of how to exploit path-diversity in the context of Application Delivery Networks (ADNs), namely the evolution of CDNs where the edge servers additionally host transactional logic for the access and manipulation of data on back-end data centers. As we will discuss, path-diversity based solutions are not trivial for Web-based transactional applications, especially in the case of transactions spanning multiple, autonomous data centers. In such a context we need in fact to deal with a set of issues not present in classical content delivery applications, such as distributed deadlocks potentially occurring for the processing of instances of a transactional request sent to multiple edge servers along different network paths. In this paper we propose a simple and lightweight path-diversity based protocol for the invocation of distributed transactions over the Web, which addresses all those issues in a scalable manner by requiring no form of coordination among the edge servers over the ADN.

1 Introduction

Over the years, a number of solutions, e.g. [5, 6, 10, 13, 15], have been proposed relying on the idea of concurrently exploiting multiple network paths among communicating parties in order to provide enhanced reliability and performance. The common base underlying these approaches is to leverage the inherent path-diversity of multi-hop networks so to reduce the likelihood to incur link congestions or failures. Specifically, in multi-hop networks, two network paths are expected to exhibit only a limited correlation, so that there is a small chance that both the paths simultaneously experience losses or failures.

Such a limited correlation expectation has been actually analyzed and confirmed for the case of Internet network paths connecting a client to a set of geographically distributed server replicas (e.g. as in the case of Content Delivery Networks - CDNs) [1, 5, 7, 11]. Among those studies, an interesting approach, based exactly on the idea of leveraging the path-diversity provided by CDNs, is the one proposed in [1]. This work has shown that significant performance benefits for the fruition of video streams can be achieved by having a pair of edge servers in the CDN simultaneously sending to a client the complementary descriptions of a video stream encoded through a multiple description coding. Thanks to the properties of this coding technique (1), disruption in streaming media occurs only in the less likely case when simultaneous losses afflict both the paths to the edge servers. More in general, a promising result highlighted in [1] is that existing CDN infrastructures seem to have the intrinsic potential for providing uncorrelated network paths among a client and multiple edge servers, even though they were originally designed to minimize distance from clients to edge servers rather than for maximizing path disjointness.

We argue that the idea of exploiting path-diversity could also be effectively employed to improve user-perceived responsiveness and availability of a wide and important class of transactional applications, whose critical nature imposes real-time constraints on the delivery of the requested results to clients. Applications such as on-line stock/equity trading, banking or auctions represent only a few remarkable examples for which the level of responsiveness and availability perceived by the end-users is an issue even more critical than in classical content delivery. Specifically, for some of those applications, reduced responsiveness/availability may translate in loss of revenue and/or customer loyalty, and may possibly give rise to legal implications. On the other hand, designing techniques leveraging path-diversity in order to improve the user perceived responsiveness/availability in transactional applications is more complex than in the case of content delivery for several reasons. This is particularly true if one considers the general scenario where the edge servers execute distributed transactions spanning multiple, autonomous data centers, as in the case of e-Commerce applications involving several

---

1This technique codes a stream into two (or more) complementary descriptions. If either description is received it can be used to decode baseline quality video, whereas both descriptions can be used to decode improved quality video [1].
In this work we present a path-diversity based protocol providing precisely those mechanisms. Specifically, we address the problem of how to exploit the replication of the edge servers, each one offering access to the transactional logic, to achieve higher levels of responsiveness and availability, while ensuring that only one among multiple contacted edge servers really executes the transactional logic on the back-end data centers. Any other contacted edge server does not even start the processing of the distributed transaction. In this way we exploit path-diversity by avoiding at all the previously pointed out problems. Actually, our solution does not require any explicit coordination among the edge servers concurrently contacted by the client application. Instead, it relies on a smart approach for the management of the connection establishment phase between the edge servers and the back-end data centers. Our solution is therefore inherently scalable, and, additionally, does not require the ADN infrastructure to provide controlled throughput and latency among the replicas of the edge server themselves. Beyond providing the description of the protocol, we also report experimental results showing minimal overhead imposed by our solution for the case of classical transaction profiles in the industry standard TPC-C benchmark [17].

The remainder of this paper is structured as follows. In Section 2 we describe the architecture of our target system. In Section 3 the protocol is introduced and discussed. In Section 4 related work is presented. Section 5 is devoted to overhead evaluation.

2 Target ADN Architecture

The architecture of our target system, i.e. the ADN, consists of a set of edge servers and a set of autonomous back-end data centers, possibly belonging to different companies. The edge servers host the business logic for executing transactions against the data centers, which are responsible for guaranteeing the availability and consistency of the application data. Furthermore, edge servers may perform some form of caching of the contents/query-results retrieved by the data centers. Actually, the interconnection between edge servers and data centers can take place either through the Internet or through a (virtual) private network under the control of the Application Service Providers (ASPs) owning the ADN. End-users access the transactional logic residing at some edge server through a client application (e.g. an applet running in a browser), which interacts with the edge servers via the Internet. The processing of the client request at the edge server consists in the execution of a distributed transaction against the data centers, and in the computation of a response message to be delivered to the client itself.

The back-end data centers perform data manipulations within ACID transactions, whose execution is modeled by the following three phases:

Connection Establishment. During the connection establishment phase an edge server contacts the data centers in order to set up a fresh transactional context for the execution of the subsequent data manipulation statements. In the following, we will model this phase through a round trip exchange of the pair of messages Connection, ConnectionACK between an edge server and a data center.

Transactional Business Logic Execution. We abstract over the details of the transactional logic (e.g. the set of SQL statements) which are obviously application dependent. Anyway, we assume that the business logic executed by the edge server consists of a transaction requiring a single or multiple interactions with the data centers, e.g. one or more stored procedure invocations.

Commit Phase. Since we consider transactions spanning multiple, autonomous data centers, an Atomic Commit Protocol (ACP) is required [8] in order to preserve the transaction atomicity (i.e. to ensure that all the data centers commit the distributed transaction, or none of them does). As the industrial standard commit protocol for distributed transaction processing is two phase commit (2PC) [16], we assume standard 2PC is employed, even though our approach is actually independent of the chosen ACP.
3 The Path-Diversity Protocol

This section is devoted to the presentation of our path-diversity protocol. For the sake of presentation simplicity, we choose to describe the behavior of the data center, before showing the client and edge server pseudo-codes. Such a choice derives from that the key mechanism employed by our protocol to address all the problems raised in the Introduction for what concerns path-diversity strategies in transactional applications over ADNs is based on the manipulation of an auxiliary data structure, called Connection Vector, which is performed by the data centers just upon the connection establishment phase. Hence knowledge of the data center behavior helps better understanding the whole protocol features.

3.1 Data Center Behavior

Figure 1 shows the pseudo-code describing the behavior of a data center during the connection establishment phase. The other phases encompassed by the transaction processing, i.e. the transaction business logic execution and the commit phase, are not explicitly described since they straightforwardly comply with conventional DBMS technology where, e.g. the 2PC protocol during the commit phase is supported by classical prepare/commit API in the standard XA programming interface [16]. Instead, as already hinted, the connection establishment phase constitutes the core of our path-diversity protocol since it embeds the mechanism used to determine which one among the contacted edge servers shall actually carry out the whole work associated with transaction processing.

As stated in Section 2, to establish a connection with a data center, an edge server sends out a Connection message and waits for a ConnectionACK. We assume that during this phase the connection messages are tagged with the identifier of the client request, namely req_id, and also with an information, namely req_inst, identifying the specific request instance, among the multiple instances possibly sent by the client to different edge servers.

The data center associates all Connection messages having same req_id value with an auxiliary data structure, which we refer to as Connection Vector for a given client request. In the following, we will use the notation CV_{req_id} to refer to the Connection Vector related to all the request instances having req_id as their identifier. CV_{req_id} is instantiated by the data center upon the arrival of the first Connection message carrying req_id as the request identifier. Each entry in CV_{req_id} maintains a value in the domain {null, refused, accepted}, and we assume all the entries are initially set to null upon the instantiation of CV_{req_id}.

Connection Vectors are accessed and manipulated by the data center within ACID local transactions, which are independent of the distributed transaction associated with the client request and coordinated by the edge servers.

By the pseudo-code in Figure 1, upon the receipt of a Connection message carrying req_id and req_inst, the data center possibly instantiates CV_{req_id} (if not already instantiated) and updates it within an ACID transaction, performing the following operations. For each distributed transaction instance x less than req_inst, all the null entries of CV_{req_id} are marked with the value refused. Then, the data center checks whether CV_{req_id}[req_inst] has null value, and, in the positive case, it updates this entry with the value accepted. Finally, the data center sends back to the edge server a ConnectionACK message tagged with req_id and req_inst, and piggybacking the vector CV_{req_id}.

**Observation.** By the pseudo-code in Figure 1, the following property trivially holds. Given the Connection Vector CV_{req_id} associated with whichever req_id, let us denote with minAccepted the minimum index for which the corresponding entry within CV_{req_id} ever had been set to the value accepted; minAccepted remains unchanged over time for that Connection Vector, independently of the receipt of any subsequent Connection message tagged with the same req_id and whichever req_inst value, possibly occurring at that data center.

3.2 Client Behavior

We show in Figure 2 the pseudo-code for the client behavior. It is quite simple since the client itself only needs to send a set of instances of a same request to a set of edge servers (for simplicity we abstract over the connection phase between the client and the edge servers, typical of standard Web-oriented interactions, through the use of message passing as the communication model). This means sending a set of Request messages tagged with the same req_id, but with different values for req_inst. Actually, the client simply uses integer values ranging from one to distinguish among the multiple request instances sent to the edge servers. After having transmitted its request instances to the selected edge servers, the client waits for a reply from one of them. Upon its arrival, the transaction result is returned to the client application.

3.3 Edge Server Behavior

Figure 3 shows the pseudo-code for the edge server behavior (single threaded for the sake of simplicity). As the first step upon the receipt of a client request, the edge server executes the connection phase to the back-end data centers.
To this purpose, it sends a `Connection` message tagged with the client request identifier and request instance number to the data centers. It then waits for `ConnectionACK` messages from all the data centers. As already mentioned in Section 3.1, the `ConnectionACK` message piggybacks the `Connection Vector` associated with that request identifier.

After having collected `ConnectionACK` messages from all the data centers, the edge server determines whether to start processing the client request or not. At this end, it checks whether the request instance number `req_inst` it is managing matches the minimum index, evaluated across all `Connection Vectors` returned by the data centers through `ConnectionACK` messages, namely `GlobalMinIndex`, for which the corresponding entry of all those vectors has value equal to `accepted`. In the positive case, the edge server starts the execution of the distributed transaction, attempts to commit it by means of the 2PC protocol and returns the result to the client. Otherwise, the edge server simply terminates (with implicit disconnection from the data centers).

Actually, by the Observation in Section 3.1, the minimum index related to an entry set to `accepted` within whichever `Connection Vector CV_{req_id}` does not vary over time. Hence, no two edge servers determine a different value for `GlobalMinIndex`. Therefore, no two different edge servers will proceed processing the distributed transaction. This is because only for one among the contacted edge servers the condition (`req_inst == GlobalMinIndex`) is verified. This is the edge server that first succeeds in updating `CV_{req_id}[req_inst]` to the value `connected` on the whole set of back-end data centers. In other words, this is the edge server that more quickly than the others completes the connection phase towards all the back-end data centers. Hence, it is the edge server that, depending on current network conditions, has the ability to support the whole end-to-end interaction associated with the client request in the most responsive manner. Note also that, whichever is the interleaving of the connection establishment requests seen by the data centers, one edge server will eventually process the client request. This depends on that for the edge server requiring connection establishment for the request with the highest `req_inst` value (this value is equal to `n` in our pseudo-code), the corresponding `req_inst` entry in the `Connection Vectors` is always updated to the value `accepted`. This ensures that there is at least one edge server for which the condition (`req_inst == GlobalMinIndex`) in line 7 of the pseudo-code in Figure 3 is verified. As a consequence, there is always an edge server that eventually processes the whole distributed transaction.

Overall, our solution allows the end-to-end interaction to be completed by letting the client request be processed only by the edge server that (depending on the current state of Internet network paths) is more responsive than all the others that have been contacted, while establishing the connection towards the data centers, whereas all the other edge servers stop processing that request just after the connection establishment phase. This allows us to effectively tackle the problems/issues we discussed in the Introduction. Specifically:

**Multiple Updates.** Since only one edge server executes the distributed transaction associated with the client request, there is no risk for multiple, undesired updates to take place at the back-end data centers.

**Additional Overhead.** The expected additional load on the edge servers and on the data centers is restricted only to the connection establishment phase which, as we will show in Section 5 by means of experimental measures, produces a minimal overhead, compared to a baseline solution not leveraging path-diversity.

**Increased Deadlock Rate.** This issue is simply circumvented by having only one transaction instance for a given client request executed against the back-end data centers despite the client contacts multiple edge servers.

### 4 Additional Related Work

As already discussed, leveraging diversity over packet networks is not a new idea. Dispersity Routing [10] and IDA [13] were probably some of the first works in this area, which proposed to split the transfer of information over multiple network paths so to provide enhanced dependability and performance. Simulation results and analytical stud-
ies have shown the benefits of this approach [2, 4] in the context of real-time communications. Compared to these results, we aim at leveraging existing ADN infrastructures to provide multiple paths without requiring explicit path-diversity support from the network.

Recent works that are close to our approach exploit the intrinsic diversity of large scale geographically distributed overlay infrastructures, e.g. CDNs, to achieve higher QoS levels in content delivery applications, such as parallel file downloads [7], cooperative Web cache sharing [9, 18] and multimedia streaming [1]. The main difference with our approach is that we deal with transactional applications, for which a set of additional difficulties (see the Introduction) need to be tackled with properly designed mechanisms (i.e. the smart connection phase between edge servers and back-end data centers in our case).

In [11], various strategies for the invocation of non-transactional replicated Web-Services were empirically evaluated and it was found out that parallel invocations have the potential for reducing the user perceived response time. One strategy evaluated in [11] is based on the idea of contacting only the more responsive among the available servers. This solution was shown to perform better than pure parallel invocation in case the response message is large with respect to the available bandwidth at the client. Our proposal intrinsically provides the same advantages since only one response message is returned to the client (i.e. the one from the edge server that really executes the whole transactional logic), hence not requiring large bandwidth at the client to reveal effective (2). Furthermore, allowing the client to choose the more responsive server requires periodic network probing, which is avoided in our approach.

Finally, in [14] a protocol leveraging path-diversity for the case of transactional applications based on a single, centralized back-end data center is proposed. In that work, the determination of which among the multiply contacted edge servers can entirely process the transaction relies on a simple primary key insertion as first action within the transaction. This mechanism is used to reject actions from edge servers that connect to the centralized data center in a non-timely manner. The difference between that work and the proposal in this paper is that we deal with the more complex and general scenario of distributed transactions spanning multiple, autonomous data centers, like for the case of e-Commerce applications involving several parties within a same business process. Here we are therefore faced with additional problems, such as distributed deadlocks of multiple instances of a same transaction, which might prevent the effectiveness of path-diversity if not adequately tackled.

2In transactional Web-based applications, request messages are typically much smaller than responses. Hence the large part of bandwidth consumption at the client is related to the response message coming from the edge server.

### 5 Evaluation

The actual improvements achievable through any path-diversity approach obviously depend on the topology of the underlying network infrastructure. Specifically, the higher is the disjointness ratio of the simultaneously explored network paths, the higher is the actual effectiveness of these approaches. A number of recent works proposing path-diversity based solutions for content delivery applications (such as parallel file downloads [7], cooperative Web cache sharing [18] and multimedia streaming [1]) have already addressed, through both simulation and empirical studies, the issue of evaluating the actual level of diversity provided by existing large scale geographically distributed overlay infrastructures, such as the ADNs we consider in this work, showing interesting and promising results. As an example, the simulation and experimental studies reported in [1], have confirmed that (i) geographical scale replication of the application access point typically provides uncorrelated network paths among a client and the selected edge servers and that (ii) the disjointness of paths between a client and its edge servers is almost symmetrical (rarely a client is located very close to, or co-located with, a server, but is typically equidistant from them). Additionally, in [11], it was found out, through an extensive empirical study, that parallel invocation of geographically replicated, non-transactional Web-Services provides up to more than 50% reduction in the user perceived response time. These results clearly indicate that ADN infrastructures are likely to achieve real user perceived benefits of the same order of magnitude also in the context of transactional applications through the exploitation of path-diversity as in our proposal. In the light of the above considerations, rather than (re)evaluating the path-diversity provided by classical ADN infrastructures or the achievable reduction in the user perceived latency, we focus in this section on estimating the overhead imposed on the data centers by our solution wrt to a baseline approach where a request instance is submitted to a single edge server.

Figure 4 shows a schematization of the behavior of the baseline and of our protocol, which performs the transactional manipulation of the Connection Vectors during the connection establishment phase. In order to evaluate the
Table 1. Quantifying the overhead per request of our approach (values in msec).

<table>
<thead>
<tr>
<th></th>
<th>Business Logic</th>
<th>Connection Vector Manipulation</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>25.44</td>
<td>3.2</td>
<td>+2.5%</td>
</tr>
<tr>
<td>PT</td>
<td>28.93</td>
<td>3.2</td>
<td>+11.06%</td>
</tr>
</tbody>
</table>

Figure 5. Data center response time vs. the request arrival rate.

overhead due to this operation, we have developed prototype implementations of modules for the transactional manipulation of the Connection Vectors, and of two transaction profiles specified by the TPC BENCHMARK\textsuperscript{T,M} C [17], namely the New-Order Transaction (NOT) and the Payment Transaction (PT). These profiles portray the activities of a wholesale supplier and are representative, respectively, of a mid-weight and of a light-weight read-write transaction. The reported measurements were obtained by running DB2/UDB v8.1 on top of Windows 2003 Server on a multi-processor machine equipped with 4 Xeon 2.2 GHz, 4 GB of RAM and 2 SCSI disks in RAID-0 configuration. We have implemented the transactional business logic through a JDBC driven stored procedure, which, being invoked locally, allows a worst case evaluation of the overhead at the data center since no network delay appears in the cost of transaction processing (recall this cost is taken as the reference point for the overhead evaluation).

For what concerns the settings for the overhead evaluation, we consider Connection Vectors supporting the case of two edge servers contacted in parallel by the client. This choice derives from that two is the typical number of edge servers that is employed by (and evaluated for) most of the proposals based on path-diversity, e.g. [1, 14]. Also, we implemented the logic performing the manipulations of the Connection Vectors by means of a JAVA module executed at the data center side, which performs local transactions on user level database tables through a standard JDBC driver. Finally, we used the classical optimization of maintaining a pool of pre-allocated database tables for the instantiation of Connection Vectors.

We report in Table 1 the average processing time for the execution of the NOT and PT TPC’s transaction profiles and, in separate columns, the average cost for manipulating the Connection Vectors at the data center side, as well as the percentage of overhead. These data clearly show that the overhead exhibited by our protocol is minimal, being around the 2% for the middle-weight NOT transaction profile, and about the 10% for the light-weight PT transaction profile. Finally, in Figure 5 we show data for the evaluation of possible variations of the system saturation point due to the additional activities required by our protocol with respect to the baseline. Specifically, we focus on evaluating the throughput reduction at the back-end data centers due to the (transactional) manipulation of the Connection Vectors. The plots in Figure 5 confirm that the limited overhead of our protocol produces very limited variation of the data center saturation point. Specifically, the anticipation in the saturation point is almost negligible for the middle-weight transaction profile (NOT) and is limited to no more than 10% of workload reduction even for the light-weight transaction profile (PT). Overall, our path-diversity protocol has the potential for reducing the end-to-end user perceived latency, while not significantly impacting resource consumption.

References