Design and Evaluation of a Parallel Invocation Protocol for Transactional Applications over the Web[*]

Paolo Romano
INESC-ID

Francesco Quaglia
DIS, Sapienza Università di Roma

Abstract

Edge computing is a powerful tool to face the challenging performance requirements of modern Internet applications. By replicating applications' data and logic across a large number of geographically distributed servers, edge computing platforms allow to achieve significant enhancements of the proximity between clients and contents, and of the system scalability. These platforms reveal highly effective when handling requests entailing *read-only* access to the application data, as these requests can be autonomously served by some edge server typically located closer to the client than the origin site. However, in contexts where end-users can trigger transactional manipulations of the application state (e.g., e-Commerce, auctions or financial applications), the corresponding update requests typically need to be re-directed to the origin transactional data sources, thus nullifying any performance benefit arising from data replication and client proximity.

In order to cope with this issue, in this article we present a parallel invocation protocol, which exploits the path-diversity along the end-to-end interaction towards the origin sites by concurrently routing transactional requests towards multiple edge servers. Request processing is finally carried out by a single edge server, adaptively selected as the most responsive one depending on current system conditions. The proposed edge server selection scheme does not require coordination among (geographically distributed) edge server instances, thus being very light and scalable. The benefits from our protocol in terms of both reduced and more predictable end-to-end latency are quantified via an extended simulation study.

Index-Terms: Path-Diversity, Transactional Applications, Web Applications, Application Delivery Networks, ASP Infrastructures, Application Level Protocols.

[*]An earlier version of this article appeared in the Proceedings of the 6th IEEE Symposium on Applications and the Internet (SAINT'06).

1 Introduction

Edge computing platforms, also known as Content and Application Delivery Networks (CDNs and ADNs) [17, 36, 39, 75], have emerged as powerful tools to face the performance challenges raised by modern Internet applications. They are based on replication of both application contents (e.g., Web objects and fragments of DBMS data) and logic (e.g., scripts for dynamic generation of Web contents and remote interaction with origin sites) across geographically distributed servers, referred to as *edge servers*, which are typically located at the border of the Web infrastructure (e.g., close to popular ISPs) so to increase the proximity between clients and contents. Such techniques are very effective in handling read-only requests, which can be autonomously served based on information locally maintained at the edge servers. On the other hand, they fail to enhance the level of service perceived by users submitting transactional update requests (e.g. validation of electronic payments in e-Commerce applications), whose processing imposes the access to the origin (primary) data sources. The problem is even more exacerbated and complex if one considers common business processes encompassing the coordination of a multiplicity of autonomous back-end services, such as those targeted by many recent Web services flow specification languages like BPEL4WS [38] and WSCI [20]. In these scenarios, the conventional approach of selecting the edge server that appears to be the most responsive while interacting with the client may deliver suboptimal performance, as it would rather be desirable to identify the edge server capable of globally minimizing the latencies towards both the client and the whole set of involved back-end data sources.

In order to cope with such a performance issue, in this article we propose a parallel invocation scheme that leverages the intrinsic redundancy and path-diversity of edge computing platforms [5, 65, 66]. This is done by routing transactional update requests towards the origin sites along multiple edge servers, and letting users perceive the latency of the fastest chain of components along the end-to-end interaction. In other words, we present an optimization strategy which allows transactional requests to be processed by the edge server that, depending on current system conditions, reveals more responsive towards both the client and the origin sites. Also, this strategy allows the system to adapt its behavior with very fine granularity, since it has the capability to select the currently best performing path (among those explored in parallel) for each single request.

We note that, over the years, a number of solutions have been proposed relying on the idea of concurrently exploiting multiple paths among communicating parties in order to provide enhanced performance. In fact, multi-path approaches have been investigated in a variety of application domains, such as packet routing at the network level (see, e.g., [18, 46, 56]), large file downloading [15, 21], cooperative caching schemes [37, 71] and overlay networks for content delivery services [4, 6, 49, 62]. On the other hand, designing parallel invocation techniques leveraging path-diversity in the context of transactional applications, such as e-Commerce, entails the following additional difficulties:

- It is not feasible to simply multi-cast a client request to different edge servers if the transactional logic is not idempotent, as this would lead to multiple, undesired manipulations (e.g. updates) at the origin sites.

- Concurrent submission of a client request to multiple edge servers results in an increase in the workload perceived by the origin sites and by the Web infrastructure, which could negatively affect the system saturation point.

- Transactions originated by different instances of the same client request, and concurrently submitted to different edge servers, are almost certainly doomed to conflict on some data sets. The combination of such a high conflict rate and the need for enforcing durable locks on updated data items (during the pre-commit phase of the standard 2PC protocol) to ensure the atomicity of distributed data manipulations, translates into an increase in the transaction deadlock (and abort) rate.

These problems point out that additional mechanisms need to be employed in order to effectively exploit parallel invocation in the context of transactional applications. These are exactly the mechanisms provided in this article. Specifically, in our protocol, update requests are processed by a single edge server among the concurrently contacted ones, which is the only one to execute the transactional logic manipulating data at the origin sites. The other contacted servers do not even start the processing of the (distributed) transaction. This simply allows us to circumvent the aforementioned problems. Further, our solution does not require any explicit coordination among the edge servers concurrently contacted by the client application. Instead, the selection of the currently best performing path relies on a smart approach for the management of the connection establishment phase between the edge servers and the origin sites. Our proposal is therefore inherently scalable, and, additionally, does not require the Web infrastructure to provide controlled throughput and latency among the edge servers.

Beyond providing the description of the proposed parallel invocation protocol, we present an extensive simulation study assessing its benefits in a wide variety of system settings. The evaluation is based on both synthetic and realistic network topologies, and relies on the TCP model in [16] to accurately simulate network latencies in normal operation mode, as well as in presence of run-time anomalies associated with, e.g., packet losses. We simulate both the scenarios of Web infrastructures layered over public networks on the Internet, and the case of Web infrastructures relying on (virtual) private interconnection between the edge servers and the back-end data centers hosting the origin DBMSs. This allows quantifying the performance benefits of our protocol when considering mainstream architectural scenarios for what concerns the organization of the edge server platform.

The remainder of this paper is structured as follows. In Section 2 we describe the architecture of our target system. In Section 3 the parallel invocation protocol is presented. Section 4 is devoted to the simulation study. In Section 5 related work is discussed.

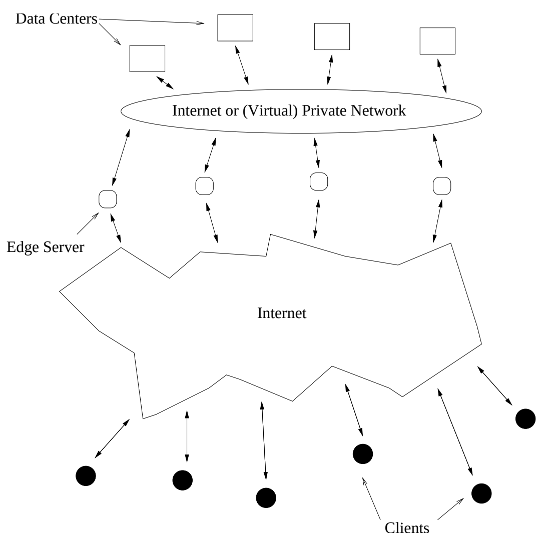


Figure 1: Target system architecture.

2 Target System Architecture

As shown in Figure 1, the architecture of our target system consists of a set of autonomous back-end data centers (the origin sites), maintaining different data sets, and of a set of replicated edge servers. Data centers are in charge of guaranteeing the availability and the transactional (ACID) consistency of the so called application "hard" state, i.e. the most critical portion of the application state, which is characterized by strict persistency requirements (e.g., a bank account balance in an e-Banking application). Edge servers provide access to the application business logic, and are in charge of maintaining (part of) the so called application "soft" state (if any), i.e., the volatile portion of the application state. In the context of Web applications, the most frequent example of soft state is the so called "session state", which tracks information about a sequence of requests from a single user within a same long-lived interaction (e.g., the content of a shopping cart prior to the checkout interaction), and which is normally discarded [25] when the user does not access the service for a while (e.g., 30 minutes). Further, according to mainstream architectural proposals [17, 36, 39, 75], edge servers typically perform caching of Web objects (either static, such as HTML pages, or dynamic, such as J2EE Servlets) and/or of fragments of (frequently accessed) DBMS tables whose primary copy resides at the back-end data centers. This is done in order to improve both scalability and client proximity to the application data.

Interconnection among edge servers and data centers normally takes place through the Internet. Alternatively, in order to ensure more controlled communication latency (at least during normal operation mode), processes may communicate on top of a (virtual) private network under the control of the Application Service Providers (ASPs) owning the whole infrastructure.

For requests entailing read-only access to the application hard state, edge servers can exploit hard state contents locally cached at the border of the network to avoid interactions with (far) back-end data centers. However, edge servers caching strategies result ineffective for client requests whose processing entails the update of the application hard state. In such a case, direct access to primary data copies of the hard state information (residing at the back-end data centers) is typically unavoidable for a number of relevant reasons, such as:

- The very same nature of the application may impose interactions with third-party services, e.g., e-Shops normally rely on external trusted authorities for validation of electronic payments.
- The de-facto standard approach [28] to transactional WAN data replication is to rely on some primary copy scheme, which imposes re-directing update requests to (remote) primary DBMSs. This is because update anywhere-anytime-anyway transactional replication strategies, such as, e.g., [3, 52, 61, 68], are known to show unstable behavior when the degree of replication and the workload scale up [28], hence revealing unattractive in large scale infrastructures.

The parallel invocation protocol we present in this paper copes with performance issues while handling requests entailing updates of the application hard state. Hence it represents a solution orthogonal to the aforementioned caching mechanisms to improve the user perceived level of service in these type of (large scale) Web infrastructures. This protocol can be particularly useful for time sensitive applications such as auctions, e-Commerce and financial trading.

3 The Parallel Invocation Protocol

In this section we first present a version of the parallel invocation protocol tailored for the case of client requests which only determine updates of the (back-end tier) hard state, without any dependence on the (edge server) soft state (i.e. the hard state update depends only on the client request parameters). This scenario is typical of applications with stateless edge servers and we will refer to it as *stateless* scenario.

Successively, we extend the protocol to cope with the treatment of requests whose effects on the application hard state depend on both the client request parameters and the current value of the soft state maintained at the edge server. In this scenario, which we refer to as *stateful* scenario, the application relies on stateful edge servers, and the soft state evolves as a function of the sequence of interactions with the client.

We note that the update of the primary copy of the application hard state, residing at the back-end data centers, actually involves the execution of a (distributed) transaction, to be handled by the edge servers. We model the execution of such a (distributed) transaction with the following two phases:

Connection Establishment. During the connection establishment phase an edge server contacts the data center(s) involved in the transaction 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 edge server and data center(s).

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, by means of the `compute_transaction` primitive. Further, we assume that the business logic executed by the edge server consists of a (distributed) transaction requiring a single or multiple interactions with the data centers.

Note that the previous transaction execution model is sufficiently general to capture both (i) the case of atomic distributed transactions, relying on distributed atomic commit protocols, such as two-phase commit (2PC) [29], and (ii) the case of distributed transactions associated with Business Activity protocols, e.g. [23], which do not exhibit atomicity requirements and are typically based on application level logic to handle business exceptions [27].

As a last preliminary observation, our protocol is not targeted to address fault-tolerance issues, but is rather aimed at exploiting redundancy and diversity on a geographical scale exclusively for performance purposes. On the other hand, fault-tolerance can be achieved via orthogonal redundancy/replication and log/restore techniques acting on a local basis (e.g. cluster-based replication techniques [22, 52, 68]) which can be used to guarantee service continuity at both the edge servers and the back-end data centers, as well as application consistency (e.g. consistency of the interaction between the edge server and the data centers [9, 10, 31, 72]) despite the occurrence of failures.

3.1 Stateless Scenario

We introduce the parallel invocation protocol for the scenario of stateless edge servers by first describing the behavior of the back-end data centers, and then the behavior of clients and edge servers. Such a choice derives from that the key mechanism employed by the protocol to address all the problems (i.e. non-idempotent transactional logic and possible increase of data center workload) associated with the parallel invocation strategy is based on the manipulation of an auxiliary data structure, which we refer to as *Connection List*. This manipulation is performed by the data centers just upon the connection establishment phase. Hence knowledge of the data center behavior helps better understanding the structure of the whole protocol.

3.1.1 Data Center Behavior

Figure 2 shows the pseudo-code describing the behavior of a data center during the connection establishment phase (the business logic execution is not explicitly described since data

```
1. wait for a Connection message with given $ReqID$ value;
2. set short timeout and wait for other Connection messages with the same $ReqID$ value;
3. upon timeout expiration:
       setup $CL_{ReqID}$;
4.     for each received Connection[$ConnectingES$,$ReqID$,$ESlist$] message:
           insert the identifier $ConnectingES$ into $CL_{ReqID}$;
5. reply with ConnectionACK[$CL_{ReqID}$] to all the edge servers in $ESlist$,
                 from which the Connection message tagged with $ReqID$ has already been received;
6  while (not received Connection[$ConnectingES$,$ReqID$,$ESlist$] message from all edge servers in $ESlist$):
7.     wait for a Connection[$ConnectingES$,$ReqID$,$ESlist$] message;
8.     if ($ConnectingES$ is greater than the maximum edge server identifier already in the $CL_{ReqID}$ list)
           insert the identifier $ConnectingES$ into $CL_{ReqID}$;
9.     reply with ConnectionACK[$CL_{ReqID}$] to $ConnectingES$;
```

Figure 2: Data Center Behavior Upon the Connection Phase.

access/manupulation modes straightforwardly comply with, e.g., conventional DBMS technology). This phase constitutes the core of our parallel invocation protocol since it embeds the basic mechanism used to dynamically determine which one among the contacted edge servers shall actually carry out the whole work associated with request (and hence 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 Connection messages are tagged with the following information:

- The identifier $ConnectingES$ of the edge server requesting connection establishment;
- The identifier $ReqID$ of the client request for which $ConnectingES$ is currently asking connection establishment;
- The list $ESlist$ of edge servers contacted in parallel by the client for this same request. With no loss of generality, we assume this list is formed by a set of distinct values among which an ordering relation (e.g. alphabetical ordering) exists.

The data center associates all the Connection messages tagged with the same $ReqID$ value with a locally maintained list, which we refer to as *Connection List* for that request identifier. In the following, we will use the notation CL_{ReqID} to refer to this data structure. Each entry in CL_{ReqID}, if any, maintains the identifier of one of the edge servers in $ESlist$. This list (i.e. its current value) is returned from the data center to the edge servers requesting connection establishment. Specifically, it is piggybacked on ConnectionACK messages. Although not explicitly stated in the pseudo-code, each ConnectionACK message also returns information about the fresh transactional context within which the (distributed) transaction associated with the client request will be executed.

By the pseudo-code in Figure 2, upon the receipt of a Connection message tagged with a given request identifier $ReqID$, the data center waits for incoming Connection messages with that same request identifier over a short timeout period (of the order of few tens of milliseconds). Afterwards, for all the received Connection messages, the corresponding edge server identifiers are inserted into the (initially empty) CL_{ReqID} list, and then ConnectionACK messages are sent back to all those edge servers from which connection establishment messages for that same client request have been already received within that timeout period.

It is possible that, within the short timeout period, Connection establishment messages did not come in at the data center from all the edge servers in $ESlist$ (recall this is the list of edge servers contacted in parallel by the client). This might depend on current system conditions, which impact both the communication delay between the client and these edge servers, and the communication delay between each of these servers and the back-end data center. For Connection messages tagged with that same $ReqID$ and arriving at the data center after the timeout has expired, the identifier of the connecting edge server is inserted into the CL_{ReqID} list only in case it is greater than the maximum edge server identifier already stored within that list. Independently of whether the insertion of a new entry within the CL_{ReqID} list takes place, this list is returned to the edge server currently requesting connection via the ConnectionACK message.

For the pseudo-code in Figure 2, there are three main observations we would like to bring to the reader attention:

Observation 1. After the statement in line 4 is executed, insertions of edge server identifiers into the CL_{ReqID} list can only occur according to a monotonically increasing rule (see line 8). Hence, given that any ConnectionACK message is ever sent out by the data center starting from line 5, all ConnectionACK messages ever sent out, in response to a Connection message tagged with a given client identifier, piggyback connection lists CL_{ReqID} whose updated values reflect the aforementioned monotonically increasing rule. In other words, if x and y are two edge server identifiers such that $x < y$, and the CL_{ReqID} list including y has already been sent out via a ConnectionACK message to whichever edge server, then no edge server will ever receive a ConnectionACK message carrying an updated value of CL_{ReqID} which also includes the edge server identifier x.

Observation 2. The edge server with the maximum identifier in $ESlist$ eventually has a corresponding entry within CL_{ReqID}. In other words, the identifier of that edge server will eventually be inserted into the CL_{ReqID} list, independently of the arrival time (within or outside the short timeout interval) of the Connection message from that edge server.

Observation 3. The identifier of any edge server in $ESlist$ for which the Connection message is delivered at the data center within the short timeout interval, is certainly recorded within the CL_{ReqID} list. We note that if a Connection message from a given edge server is delivered at the data center within that interval (or even triggers the beginning of the timeout interval, being it the first Connection message tagged with that request identifier), it means that this edge server is one of those edge servers in *ESlist* which, depending on current system conditions, more timely than others (i) are reached by the client request and, in turn, (ii) reach that data center. Hence, when focusing on that data center, this edge server is a good candidate for dynamic optimization of the performance of the whole end-to-end interaction associated with the client request.

Practicality Issues. In practical life, our solution to handle the connection phase through auxiliary Connection Lists can be supported without requiring any modification in the inner logic of core software components at the data center, such as the hosted DBMS. Specifically, the logic described by the pseudo-code in Figure 2 can be implemented by means of a stub wrapping the connections between the edge server and the database server at the data center. Concerning this point, we also note that, although for each client request the data center receives multiple connection messages (just due to the parallel invocation scheme), a unique fresh transactional context can be set up for that client request via the wrapping approach, and communicated to all the connecting edge servers. This context will be finally used by the unique edge server that is selected for request/transaction processing.

3.1.2 Client Behavior

We show in Figure 3 the pseudo-code for the client behavior. It is quite simple since the client only needs to send Request messages in parallel to the set of edge servers in *ESlist*, tagged with the request identifier $ReqID$ (obtained by appending the sequentially increasing integer value $Timestamp$ to the unique client identifier $ClientID$), and with the request content Req ([1]). After having transmitted its request in parallel to the selected edge servers, the client waits for a reply from one of them. (As it will be clear after the explanation of the edge server pseudo-code, only one edge server will reply. This is the unique edge server finally taking care of processing the - distributed - transaction associated with the client request.) Next, the client increases the $Timestamp$ and returns the received result.

Practicality Issues. In conventional technology, a number of viable alternatives exist for implementing the protocol logic described in Figure 3 within a client stub, such as an applet running in a Web browser or an ad-hoc coded Web-Service client.

For what concerns the issue of providing the client stub with the network addresses of a conveniently located (if not optimally located) set of edge servers (i.e. the elements in *ESlist* - see line 1 of the pseudo-code in Figure 3), which minimize the distance from the client and maximize the path disjointness), several approaches are feasible. Interested readers can refer to the works in [11, 12, 44, 69] which propose a number of techniques for addressing this issue. A possible approach, suggested in [11], is to determine the convenient set of edge servers for

[1]We abstract over the transport-layer 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.

```
1.  $ESlist = \{ES_1, \ldots, ES_n\}$;
2.  $ResultType\ Res$;
3.  $RequestType\ Req$
4.  $RequestIdentifier\ ReqID$;
5.  $ClientIdentifier\ ClientID = getUniqueClientID()$;
6.  $Integer\ Timestamp = 0$;

7.  ResultType issueRequest($RequestContent\ Req$) {
8.      $ReqID$=concatenate $ClientID$ and $Timestamp$;
9.      send Request[$ReqID$,$Req$,$ESlist$] to each edge server in $ESlist$;
10.     wait for a Response[$Res$] message from any edge server in $ESlist$;
11.     $Timestamp{+}{+}$;
12.     return $Res$;
13. }
```

Figure 3: Client Behavior.

a given client as a function of the client network address. In this case, the information about the convenient set of edge servers could be made available at the client stub, e.g., via an apposite service, or at download time in case the client stub is implemented as an applet.

Regarding the selection of a unique client identifier (see line 5 of the pseudo-code in Figure 3), which is required to uniquely tag request messages for correct association with Connection Lists maintained at the data centers, this could be implemented in a variety of practical ways. In case the client stub is implemented as an applet, the simplest approach is to let the edge server initialize the applet, at download time, with an identifier UID uniquely associated with the applet itself (e.g. by using the naming rules specified for OSI CCR atomic action identifiers [51]). Hence, the applet could even generate a unique identifier for a set of client instances by simply attaching to its own UID, a monotonically increasing sequence number.

As a final note, even though the client pseudo-code prescribes that each request is sent in parallel towards multiple edge servers, in practical settings, it could be possible to trigger the activation of the parallel invocation scheme at the client side on a periodic basis, where the period could be adapted on the basis of, e.g., network stability periods. (If the end-to-end interaction is repeatedly faster via a same edge server among those contacted in parallel, the client can temporarily exclude all the destination servers but that one while sending out its requests. On the other hand, the excluded edge servers could be re-included in the destination set for the parallel invocation scheme in case performance degradation in the interaction with the originally faster edge server is perceived.)

3.1.3 Edge Server Behavior

Figure 4 shows the pseudo-code for the edge server behavior (single threaded for the sake of simplicity). As the first step upon the receipt of the client request, the edge server determines

```
1.  $DataCenterList\ DClist$;
2.  $ResultType\ Res$;
3.  $EdgeServerIdentifier\ ConnectingES$;

4.  while ($true$)
5.          wait for a Request[$ReqID$,$Req$,$ESlist$] message from client;
6.          set $ConnectingES = GetMyID()$;
7.          determine from $Req$ the list of data centers $DClist$ involved in the request;
8.          send Connection[$ConnectingES$,$ReqID$,$ESlist$] to each $DC_i \in DClist$;
9.          wait for ConnectionACK[$CL^i_{ReqID}$] from each $DC_i \in DClist$;
10.         if ( $ConnectingES = min(es \in ESList\ s.t.\ \forall DC_i \in DCList,\ es \in CL^i_{ReqID})$ )
11.             $res$=compute_transaction($req$);
12.             send Response[$Res$] to client;
```

Figure 4: Edge Server Behavior.

the back-end data centers involved in the request (which are used to populate *DClist*), and executes the connection phase towards them. To this end, the edge server sends to each of these data centers a Connection message tagged with (i) its identifier, (ii) the request identifier, and (iii) the list of edge servers contacted in parallel by the client. It then waits for ConnectionACK messages from all the data centers. As already mentioned in Section 3.1.1, each ConnectionACK message from the data center DC_i piggybacks the Connection List locally maintained by that data center for that request identifier. We denote as CL^i_{ReqID} the list piggybacked by the ConnectionACK message from DC_i.

After having collected ConnectionACK messages from all the data centers, the edge server determines whether to start processing the client request or not. To this end, it checks (see line 9 of the pseudo-code in Figure 4) whether its identifier *ConnectingES* (retrieved via $GetMyID()$) corresponds to the minimum value in *ESlist* which is recorded within all the CL^i_{ReqID} Connection Lists received via ConnectionACK messages from the data centers. More formally, the edge server verifies whether the following condition holds:

$$ConnectingES = min(es \in ESlist\ s.t.\ \forall DC_i \in DClist,\ es \in CL^i_{ReqID}) \qquad (1)$$

In the positive case, the edge server starts the execution of the (distributed) transaction on the back-end data centers, computes the result message and sends this message back to the client. Otherwise, processing of the client request at that edge server simply stops (with implicit disconnection from the data centers).

Practicality Issues. Edge servers, and more in general application servers, exhibit a strong trend to be implemented on top of off-the-shelf industry middleware frameworks (e.g. Sun Microsystems' Java 2 Platform Enterprise Edition - J2EE, and Microsoft .NET). In addition to providing an integrated environment for component execution, which significantly reduces the time to design, implement, and deploy applications, such frameworks incorporate "best practices" design. The latter provides developers with design patterns, suggesting a standardized structure distributed component-based systems should be based on [43]. For what concerns transaction management, industry standard middleware layers already embed ad-hoc services, such as the well-known JTS for the J2EE platform, providing the abstraction of "Container Managed Transactions" [63]. This approach aims at saving application developers from the burden of implementing low level mechanisms, e.g. transaction demarcation, coordination and connection establishment. In such a context, the edge server logic we propose for handling the smart connection phase, based on Connection Lists, could be delegated to the container, whose behavior could be controlled in a simple declarative way by means of extended deployment descriptors.

3.2 Performance Optimization Capabilities

In this section we highlight how the edge server selection logic at the heart of our protocol allows identifying the edge server that is more likely to ensure the best end-to-end latency in the processing of client requests.

By **Observation 3** in Section 3.1.1, if the identifier of an edge server is recorded within whichever CL^i_{ReqID} list during the short timeout interval, then that edge server is a good candidate for efficiently supporting the whole end-to-end interaction towards all the back-end data centers. In other words, depending on current system conditions, that edge server is expected to provide high responsiveness, and hence optimized performance, while handling the client request. On other hand, if an edge server experiences high latencies while connecting to a *subset* of the involved data centers, its Connection messages are likely not to be received within the timeout interval at these data centers. This reduces the likelihood for the corresponding edge server identifier to be successfully inserted into the CL^i_{ReqID} lists locally maintained by these data centers. As a consequence, there is a reduced probability that such an edge server finds the condition in expression (1) verified and that it is selected for finalizing the processing of the client request. Hence, our protocol addresses the issue of dynamically inhibiting processing activities at the edge servers experiencing large variance in the communication delay towards different data centers, which might negatively impact the user-perceived latency (and its predictability). This is done via self-exclusion mechanisms (not requiring explicit distributed coordination) according to which these edge servers simply abandon processing activities in favor of a server that results highly responsive towards the client and all the involved data centers. Finally, such a dynamic determination of a suited edge server can be performed an a per-request basis, thus allowing the system to optimize its behavior with very fine granularity.

3.3 Correctness Arguments

In this section, we analyze how our protocol ensures exactly-once semantic for the processing of client requests, thus avoiding situations where none of the edge servers processes the client request because of the employed self-exclusion mechanism, and where multiple-updates occur due to parallel request transmissions. To this end, we develop two key correctness arguments. First we show that our scheme guarantees *at-most-once semantic*, namely that it is impossible for any two edge servers to activate the processing activities for a same client request. Then we show that it guarantees *at-least-once semantic*, namely that the self-exclusion mechanism can never prevent *all* of the edge servers contacted by a client from activating the processing of its requests ([2]).

At-most-once semantic. By **Observation 1** in Section 3.1.1 (i.e. by the monotonically increasing rule for the update of CL^i_{ReqID} lists), no two different edge servers will both find the condition in expression (1) verified. Therefore, no two different edge servers will both proceed processing the client request and the associated (distributed) transaction on the back-end data center(s). This ensures at-most-once semantic for the processing of the client request, while also limiting the additional load on the edge servers and on the data centers even in the presence of parallel invocations from the client (this is because additional tasks due to parallel invocations are restricted only to the connection establishment phase).

At-least-once semantic. By **Observation 2** in Section 3.1.1, there is at least one edge server (i.e. the one with the maximum identifier in $ESlist$) whose identifier is eventually inserted within every CL^i_{ReqID} list. Hence there is at least one edge server for which the condition in expression (1) is eventually verified. This ensures at-least-once semantic for the processing of the client request, which, together with the aforementioned at-most-once semantic, ultimately guarantees exactly-once semantic. Hence, we avoid at all the problem of multiple updates at the back-end data centers due to non-idempotent transactional logic even in the presence of the parallel invocation strategy.

3.4 Stateful Scenario

In this section we extend, in a modular fashion, the protocol presented in Section 3.1 to cope with the scenario of applications relying on stateful edge servers. As previously discussed (see Section 2), in this scenario the business logic depends on and affects both the hard state residing at the back-end data centers, and the soft state locally maintained by the edge servers. In the following, we assume that the edge servers' soft state is not shared among different clients. Hence, processing activities of requests associated with different clients entail (read/write) access to distinct portions of the local soft state maintained by the edge server. Such an assumption very commonly holds in real-world Web applications, where the middle-tier soft state (which is also referred to as "session state") is normally stored in the

[2]Note that, as already highlighted in Section 3, we do not take explicitly into account issues associated with the occurrence of failures, which we assume masked by complementary fault-tolerance mechanisms. Hence, arguments supporting exactly-once processing semantic are in this section intended as a mean to establish intrinsic properties associated with our protocol logic.

```
1.   $DataCenterList\ DClist$;
2.   $ResultType\ Res$;
3.   $EdgeServerIdentifier\ ConnectingES = GetMyID()$;
4.   $IntegerList\ NextTimestamp = \{0, \ldots, 0\}$;
5.   $SoftStateTypeList\ SoftState = \{null, \ldots, null\}$;
6.   $SoftStateType\ SoftStateUpdate$;

7.   while ($true$)
8.       wait receive Request[$ReqID$,$Req$,$ESlist$] message from client $ClientID$
             where $getTimestamp(ReqID) = NextTimestamp[ClientID]$;
9.       determine from $Req$ the list of data centers $DClist$ involved in the request;
10.      send Connection[$ConnectingES$,$ReqID$,$ESlist$] to each $DC_i \in DClist$;
11.      wait receive ConnectionACK[$CL^i_{ReqID}$] from each $DC_i \in DClist$;
12.      if ( $ConnectingES = min(es \in ESList\ s.t.\ \forall DC_i \in DCList,\ es \in CL^i_{ReqID})$ )
             // execute the business logic
13.          ($res$,$SoftState[ClientID]$)=compute_transaction($req$,$SoftState[ClientID]$);
14.          send Response[$Res$] to client;
15.          send StateUpdate[$ClientID$,$nextTimestamp[ClientID]$,$SoftState[ClientID]$]
                 to each $es \in ESList$;
16.      else // wait for soft state update
17.          wait receive StateUpdate[$ClientID$,$nextTimestamp[ClientID]$,$SoftStateUpdate$]
                 from any $es \in ESList$;
18.          $SoftState[ClientID] = SoftStateUpdate$;
19.      $NextTimestamp[ClientID]$++;
```

Figure 5: Edge Server Behavior (Stateful Scenario).

form of a per-client associative array, whereas management of any shared application state is delegated to the back-end data centers, ensuring its consistency through the notion of (ACID) transactions.

For the stateful scenario, the client and the data center behavior simply inherit the pseudo-code provided in Figure 3 and Figure 2. Conversely, the edge server behavior has to be modified in order to ensure the following two additional properties:

P.1 The evolution of the soft state trajectory must be the same for the whole set of edge servers contacted in parallel by the client.

P.2 If an edge server activates the execution of the business logic for a given client request (via the execution of the `compute_transaction` primitive), then its local soft state needs to consistently reflect the effects of any request previously submitted by that same client.

The pseudo-code describing the behavior of the edge server is reported in Figure 5 (again single threaded for the sake of presentation simplicity). We model the presence of the application soft state at the edge server by means of the *SoftState* array, which we assume indexed on the basis of *ClientID* values, and initialized with the default *null* value. Additionally, the edge server maintains an array of integers, *NextTimestamp*, indexed again via *ClientID* values, whose entries (initialized with the value zero) are used to keep track of the latest timestamp of requests received from each client, on whose behalf the edge server is storing the soft state. Manipulations of the soft state are captured by the execution of the `compute_transaction` primitive, which takes as input parameter, in addition to the request content, the soft-state associated with the client whose request has to be processed. We assume that the `compute_transaction` primitive returns, along with the result to be delivered to the client, also the updated value of the soft state associated with that client.

The logic for the selection of the edge server that really executes the business logic, among those contacted in parallel by the client, is the same as the one previously presented for the stateless scenario (see Section 3.1.3). However, in this case, the edge server dynamically selected for processing the standing request also activates a state propagation mechanism aimed at ensuring that every other edge server (contacted in parallel by the client) updates the corresponding soft state with the latest state version, which has been produced by the execution of the `compute_transaction` primitive. This is done via a broadcast operation of the StateUpdate message to every other edge server in *ESList* (see line 15). This message is tagged with the *timestamp* associated with the client request, and carries the updated soft state version. All the edge servers in *ESlist* that have not been selected for processing the standing request remain waiting for the StateUpdate message (in order to install the update state value - see lines 17-18) before accepting other requests from that same client. Finally, the edge server updates the corresponding entry within the *NextTimestamp* array, independently of whether it has executed the business logic or not (see line 19).

Note that, upon the receipt of the request message in line 8, the edge server performs an additional check to verify whether the request timestamp (extracted by means of the `getTimestamp` primitive) coincides with the expected one, locally stored in the *nextTimestamp* array. This, together with the simple above described state synchronization mechanism, ensures that request messages from a specific client, as well as the corresponding state updates, are processed in the same order at each edge server contacted within the parallel invocation scheme. This prevents divergences of the edge servers' copies of the soft state associated with that client, hence guaranteeing property P.1. Further, the proposed state synchronization mechanism also ensures that if an edge server starts processing the client request, then any soft state update associated with the execution of any previous request from that same client (i.e. request messages having lower timestamps) has already been locally installed, hence ensuring property P.2.

Given that the broadcast phase of the StateUpdate message (and the related state installation operation) is overlapped with the result communication phase towards the client, and with any think time between successive requests from that same client, in practical settings no direct impact on the end-user perceived latency is expected due to the soft state synchronization mechanism ([3]). Hence, dynamic adaptation to current system conditions in the stateful scenario still has the capability to operate with per-request granularity. This is because all the edge servers in *ESlist* are expected to have already realigned their soft state when the next request message from a given client arrives.

3.5 Remarks

As mentioned in Section 1, edge server platforms make extensive usage of caching techniques, that are de facto orthogonal to our protocol. In fact, our proposal copes with client requests updating the application state, while the aforementioned caching techniques replicate static and dynamic contents (including DBMS data fragments) across the edge servers to enhance the performance of read-only requests.

As widely demonstrated by characterizations of well known benchmarks for Web applications (e.g. the TPC-W benchmark [67]), update requests typically represent about the 10% of the whole volume of client interactions. Hence, our parallel invocation protocol is expected not to (significantly) impact cache hits at the edge servers (depending on the eventually selected edge server among the multiple contacted ones for an update request) since actually cached data are mostly determined by read-only requests, representing the largest percentage of the workload.

Also, from the perspective of individual clients, stability of cached contents is likely guaranteed since transactional update requests are usually submitted only as the concluding step of a sequence of interactions (e.g. the final submission of a purchase order after a browsing session in an e-Shop), whose outcome delivery at the client side is time-critical (since it might trigger the activation of, e.g., some transactional billing logic) and can be optimized via our proposal.

4 Simulation Study

This section is devoted to a quantitative study of the effects of the proposed protocol. While it would be ideal to deploy the protocol and evaluate it on top of real ADN infrastructures, these platforms (e.g. the one by Akamai) are proprietary and not publicly accessible. On the other hand, distributed platforms available for research experimentation, such as Planetlab [55], are characterized by arbitrary (not ADN-like) network topologies. Therefore, we choose to base our analysis on a detailed simulation model ([4]). This is actually the same approach taken in previous quantitative studies (see, e.g., [6]) targeting multi-path protocols in the context of (overlay) infrastructures supporting Web applications.

The study is focused on the quantification of the benefits achievable through our proposal

[3]We note that processing and communication latencies for state update notification/installation can be kept very low via incremental techniques [54].

[4]The simulation data reported in this section are also made publicly available for download at the URL: `http://www.dis.uniroma1.it/~romanop/files/TOC-SI-DATA.zip`

in terms of user perceived latency, and is not aimed at evaluating the potential effects of additional resources' consumption, possibly arising at high system load. This choice is motivated by the following observations, jointly pointing out minimal expected overhead associated with the protocol:

- Despite the concurrent submission of multiple client requests, the business logic is executed by a single edge server, as in a classical scenario not exploiting parallel invocation.
- As shown in previous studies related to content delivery applications [5, 6], the number of paths that is expected to maximize the performance benefits from a parallel invocation protocol is of the order of two. Conforming to such a degree of "ideal" parallelism, the operations associated with the propagation of soft state updates, required by our protocol in the stateful scenario, are in practice limited to the delivery of a single state update message per client-initiated interaction. The corresponding overhead is therefore expected to be very small, especially if one relies on efficient, incremental state update mechanisms [54].
- Given that in our protocol a single response message is returned to the client, with the recommended degree of parallelism of the order of two, we have that the additional bandwidth consumption at the client side is only due to the send of an additional request message. Since in (transactional) Web-based business applications, request messages are typically very small [45], it follows that the additional bandwidth consumption at the client side is expected to be very limited.
- Compared to a baseline approach not employing parallel invocation, the only extra work imposed by our protocol at the data center is the manipulation of the Connection List. Considering that, as hinted above, in practical settings the employed parallelism level is expected to be relatively low, Connection Lists are expected to be small size data structures. Consequently, the overhead associated with their manipulation is in practice negligible.

As a last preliminary observation, we recall that, for the stateful scenario, the edge server soft state (incremental) propagation mechanism occurs in "background", thus overlapping with result delivery at the client side and user think-time in between two subsequent requests. Given that user think-times in transactional Web applications have expected values of several secs (the well-known TPC-W benchmark [67], e.g., specifies an average think time of 7 seconds), such a background phase is very likely to be completed before the next request is issued by the client.

On the basis of the above observation, in the stateful scenario, the only operation whose execution cost additionally contributes to the end-to-end client perceived latency (compared to the stateless case) is the soft state update locally performed by the application server eventually selected by our parallel invocation protocol. However this cost is not directly imputable to our protocol, but is intrinsic to the stateful programming model. As a consequence, we argue that a detailed simulation study related to the effects of our protocol in the stateless context is representative of the performance benefits achievable in general contexts (i.e. stateless vs stateful). This is the reason why we focus on the stateless case in this simulation study.

4.1 Network Model

As highlighted in previous studies [4, 6], the effectiveness of any parallel invocation solution strongly depends on the actual disjointness among the simultaneously explored paths. To determine how our proposal fares in different networks, we took an approach similar to the one used in [6]. In our experiments, we examined both Brite [48] generated topologies (in this case both flat and hierarchical topologies have been considered, which we will refer to as BRITE-f and BRITE-h, respectively) and the NLANR [50] graph, representative of connectivity among Internet autonomous systems at the latest available date, namely January 2000.

To assign the client, edge server and data center roles to a subset of the nodes in the topologies, we used a placement algorithm based on the connectivity degree of nodes, analogous to the one employed in [6]:

- *Edge Servers:* To emulate edge server location in ADN infrastructures, we placed edge servers at the edges of a topology, where edges are defined as nodes with connectivity degree of two or three.
- *Data Centers:* To emulate data center location at the most connected part of a network, we placed data centers at the core nodes of the topology, which we define as nodes with the highest connectivity degrees.
- *Clients:* To emulate client location at the furthest edge of a topology, clients were randomly chosen among those nodes having connectivity degree of one.

To generate realistic values for the network latencies perceived by the hosts participating in our protocol, under both normal and anomalous (e.g. congested) situations, the considered topologies were complemented by both mathematical models and publicly available empirical measurements of Internet latencies. For what concerns the packet loss model across the links, we chose the widely adopted two-state Gilbert model parameterized by transition probabilities $\{p,q\}$ where p is the probability of going from no loss state to loss state, and q is the probability of going from loss to no loss. The Gilbert model is widely used to capture burst traffic for its simplicity and mathematical tractability. Like in several other studies, e.g. [6], we assumed for simplicity that faults over each link can be modelled as independent.

In order to accurately determine the message transfer time over TCP connections in presence of packet losses, we adopted the TCP analytical model in [16]. This model provides accurate estimations of TCP transfer times on the basis of (i) the number of TCP fragments to be sent (i.e. the message size), (ii) the expected number of packet losses, and (iii) the end-to-end RTT latency. Given that a number of studies (e.g. [8]) have shown that WWW traffic exhibits heavy-tailed message size distributions, our simulator determines the message size according to a Pareto distribution. The end-to-end RTT for each message transmission is derived by means of the RTT probability distribution shown in [2], that was empirically obtained via RTT measurements carried out between the NASA Glenn Research Center Web server and its clients. These RTTs are representative of end-to-end network latency between hosts communicating across the Internet. In order to correlate the length (in terms of number of hops) of a path in a topology with the corresponding end-to-end RTT value, we determined the RTT on each link over which packets are transmitted by scaling (dividing) the end-to-end value by the average path length.

Note that in practice a strong correlation exists between the link RTT and the occurrence of packet losses over that link. In fact, RTT values are comprehensive of router queueing delays, which are likely to be large in case of packet losses (since losses are typically due to the excessive growth of router queues). In order to capture such a correlation in our simulator, in absence of packet losses we randomly pick the current link RTT from the first half of the empirical RTT distribution, namely the half collecting the lowest measured RTT values. Conversely, in presence of packet losses over a link, we randomly pick the current link RTT from the second half of the empirical RTT distribution.

As a final observation, the employed network model assumes that the additional network load due to the usage of multiple paths (instead of a single path) has negligible impact on network behavior (e.g. on the packet loss rate).

4.2 Edge Server Selection Policies

In Web infrastructures not leveraging path diversity, client requests are routed towards a single edge server over a single path, and the selected edge server is typically the one on the shortest path to the client. This mechanism may be straightforwardly adopted in our proposal by selecting the closest edge servers to the client, or one may rely on more sophisticated policies, e.g. [34, 76], taking into account specific topological information in order to achieve larger benefits from the multi-path approach.

To cope with a relatively wide spectrum of possibilities, we implemented the following three selection policies in our simulator:

- *Shortest Paths.* Simply choose the closest edge servers to the client, employing hop counts as distance metric. In the following, we will refer this selection policy to as SP.
- *Disjointness Ordered Paths.* Always select the edge server on the shortest path. Then choose the edge servers whose paths towards the client have a minimum number of links in common with the shortest path. If more than one server has the same number of joint links with the shortest path, choose the one having minimum length (measured in hop counts). In the following, we will refer this selection policy to as DP.
- *Disjointness×Length Ordered Paths.* Always select the edge server on the shortest path. Then choose the edge servers whose paths have the minimum values of the product

			avg. path length client ↔ edge server			avg. path length edge server ↔ data center			avg. path correlation client ↔ edge server		
Topology	#nodes	#edges	SP	DP	D×LP	SP	DP	D×LP	SP	DP	D×LP
BRITE-f	5000	5000	9.1	9.3	9.1	8.9	8.9	8.9	0.53	0.46	0.52
BRITE-h	5000	5100	15.1	15.9	15.3	25.6	25.6	25.6	0.30	0.16	0.22
NLANR	6474	24467	3.0	3.1	3.0	2.3	2.3	2.3	0.42	0.35	0.41

Table 1: Summary of Topological Parameters.

between (i) the correlation with the shortest path and (ii) the additional length with respect to the shortest path. With this policy, if the path towards an edge server is highly disjoint from the shortest path, but such edge server is very far from the client, this edge server will not be considered by the client as a good candidate for the parallel invocation scheme. In the following, we will refer this selection policy to as D×LP.

For the reader convenience, we report in Table 1 a summary of the main parameters related to the different analyzed network topologies, together with information on the length and correlation of network paths for the different edge server selection policies (i.e. SP, DP and D×LP).

4.3 Data Center Model

In order to model the activities of the data centers, we have integrated in our simulation framework a detailed distributed database system simulator that was previously used in a number of “DBMS-centric” simulation studies, e.g. [33, 64, 73]. To realistically set the parameters defining the behavior of the aforementioned DBMS simulator, the model of the transactional business logic has been derived from the workload characterization of the TPC-W benchmark [67] that has been presented in [42] .

4.4 Simulation Parameters Settings

Edge Server Platform. The results presented in the following section were obtained considering an infrastructure consisting of 500 clients, 6 back-end data centers and 20 edge servers. We focused on the case of two edge servers contacted in parallel by the client. In fact, as already highlighted, previous studies [5, 6] have shown that the additional benefits arising from further increasing the parallelism level (beyond two) are negligible, i.e. “two” represents a point of diminishing return for what concerns the number of paths to simultaneously explore in a path-diversity protocol.

Network Links. To capture network congestion/overload situations, we have set the parameter q of the Gilbert model to the value 0.8, which corresponds to an expected burst loss length of 1.25. (It has been shown [70] that consecutive losses rarely last more than four packets and the value q=0.8 corresponds to the longest average burst length measurement we are aware of.) For what concerns the parameter p, we have considered two different values in the simulation study, selected as representative of interconnection between edge serves and data centers either via Internet or via a (virtual) private network under the control of the ASPs. In the former case, p was set to yield a moderate end-to-end loss rate of 5% for an average path length of 3 to 16 hops, depending on the topology. In the latter case, p was set to yield the extremely reduced end-to-end loss rate of 1% for the same average path lengths. The message size distribution has been obtained through a Pareto with α=1.5 and b=2.

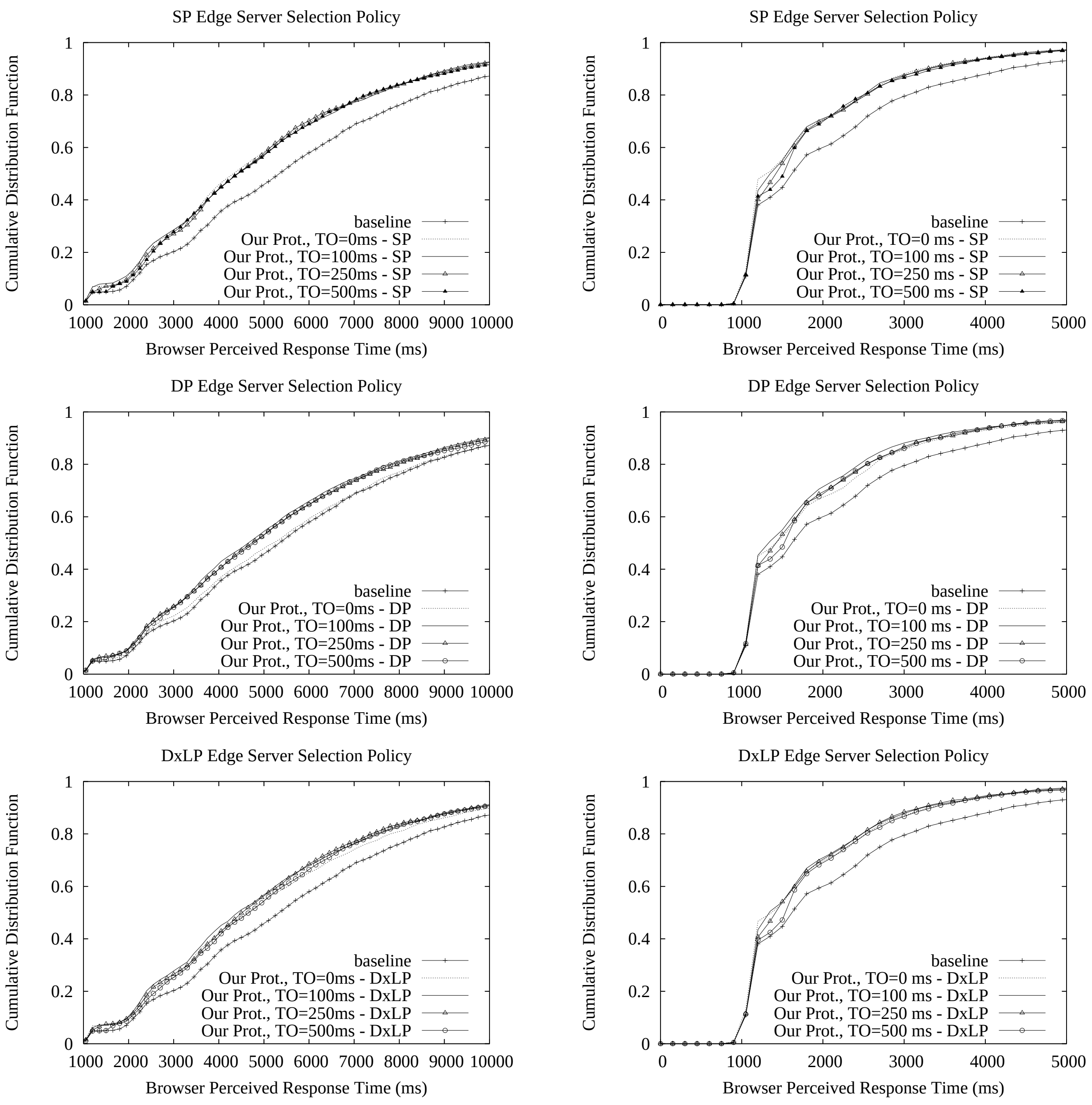
Data Centers. We focused on the TPC-W "shopping" transaction profile, which specifies a transactional mix composed at the 80% by read-only (e.g. product browsing) interactions and at the 20% by update interactions (e.g. product ordering). According to the data in [42], when considering a standard 20 GB data set, the TPC-W shopping profile gives rise to an average number of 35 referenced pages (with 4KB pages) per transaction, with 96.6% of page references in read-only mode, and 3.4% of page references in write mode. The simulator parameters describing the hardware resources available at data centers have been set to model the platform used in the workload characterization study in [42], namely an IBM eServer xSeries 255 machine, with 4 CPUs (1.5 GHz), 8 GB of RAM storage, 12 IBM U320 disks (15000 RPM), with the DBMS placed on a 5-disk hardware RAID-0.

4.5 Results

We report in Figure 6, Figure 7 and Figure 8, the cumulative distribution function (CDF) of browser perceived response times for the two considered Brite topologies (flat and hierarchical) and for the NLANR topology. In other words, we report on the Y-axis the experimentally evaluated probability for a browser to experience response time lower than the corresponding value on the X-axis.

The plots report results for both a baseline protocol not employing path-diversity (where the only edge server contacted by the client is selected according to the shortest path policy) and our parallel invocation protocol (with the three different policies for selecting the edge servers to be contacted in parallel by the client). To accurately quantify the performance advantages for requests updating the application hard state, the collected statistics do not take into account latency samples associated with read-only requests. For our protocol, we have also varied the value of the timeout (TO) used at the data centers during the connection phase in the interval between 0 and 500 milliseconds.

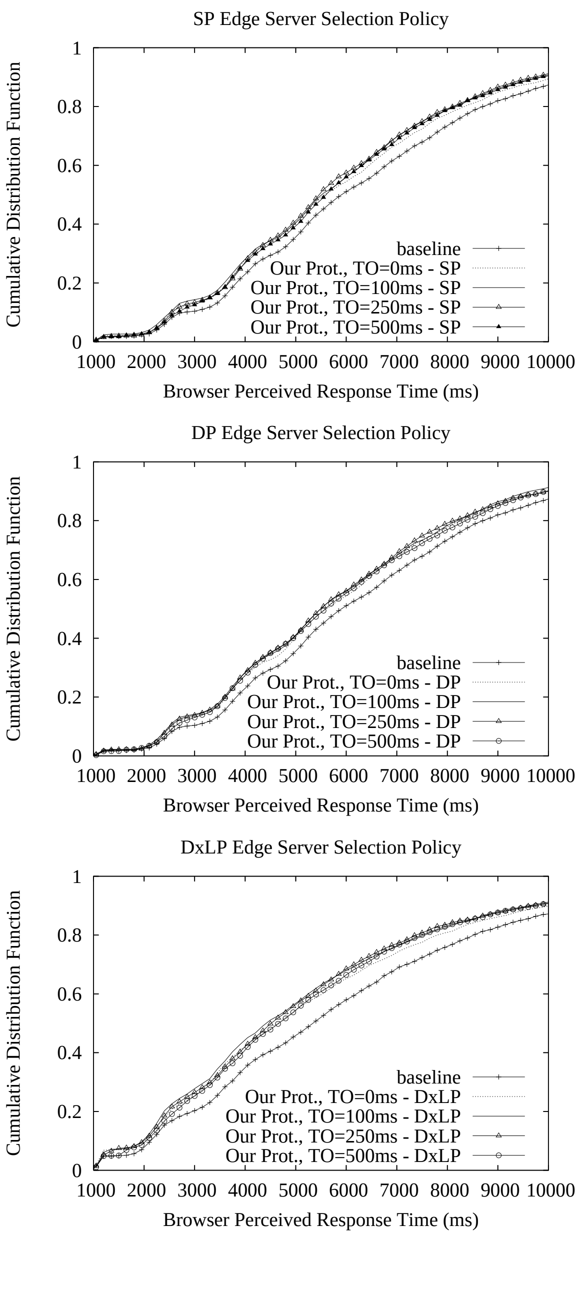
By the plots we get that the parallel invocation protocol provides remarkable benefits, in terms of increased system responsiveness. For the case of edge servers communicating with data centers via the Internet, exploiting parallel invocation in the BRITE-f and NLANR topologies (see Figure 6(a) and Figure 8(a)) allows achieving browser perceived response times less than 7 seconds (i.e. less than the maximum value complying with a reasonable expectation for an interactive end-user [78]) in about the 80% of the cases, whereas the baseline protocol achieves response times less than 7 seconds in about the 65% of the cases. (Slightly reduced benefits are provided by the parallel invocation approach when the DP edge server selection policy is employed. This is due to the fact that the alternative path selected by DP - in order to maximize disjointness wrt the shortest path - might be significantly



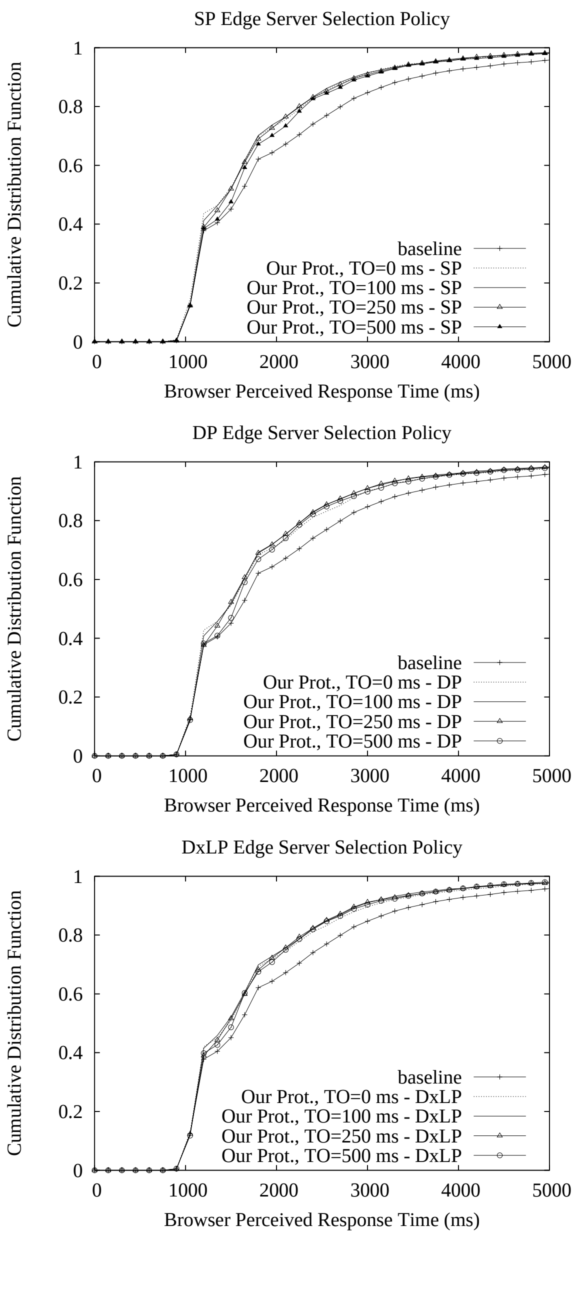
(a) Public Network
Connecting Edge Servers and Data Centers

(b) (Virtual) Private Network
Connecting Edge Servers and Data Centers

Figure 6: Browser Perceived Response Time CDF - Brite Flat Topology.

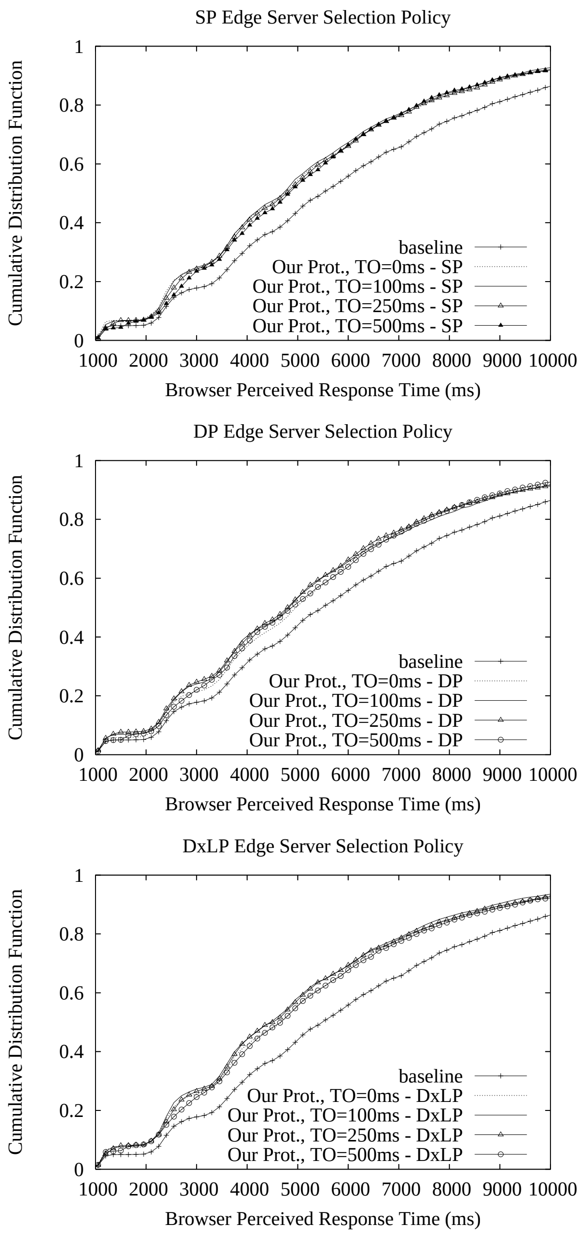


(a) Public Network
Connecting Edge Servers and Data Centers

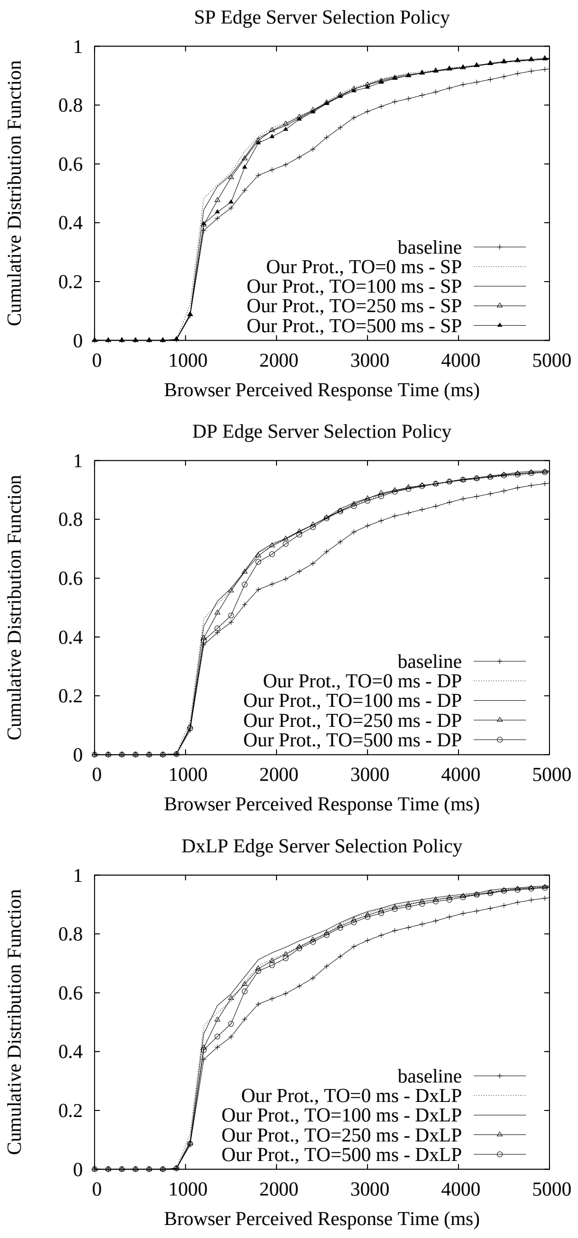


(b) (Virtual) Private Network
Connecting Edge Servers and Data Centers

Figure 7: Browser Perceived Response Time CDF - Brite Hierarchical Topology.

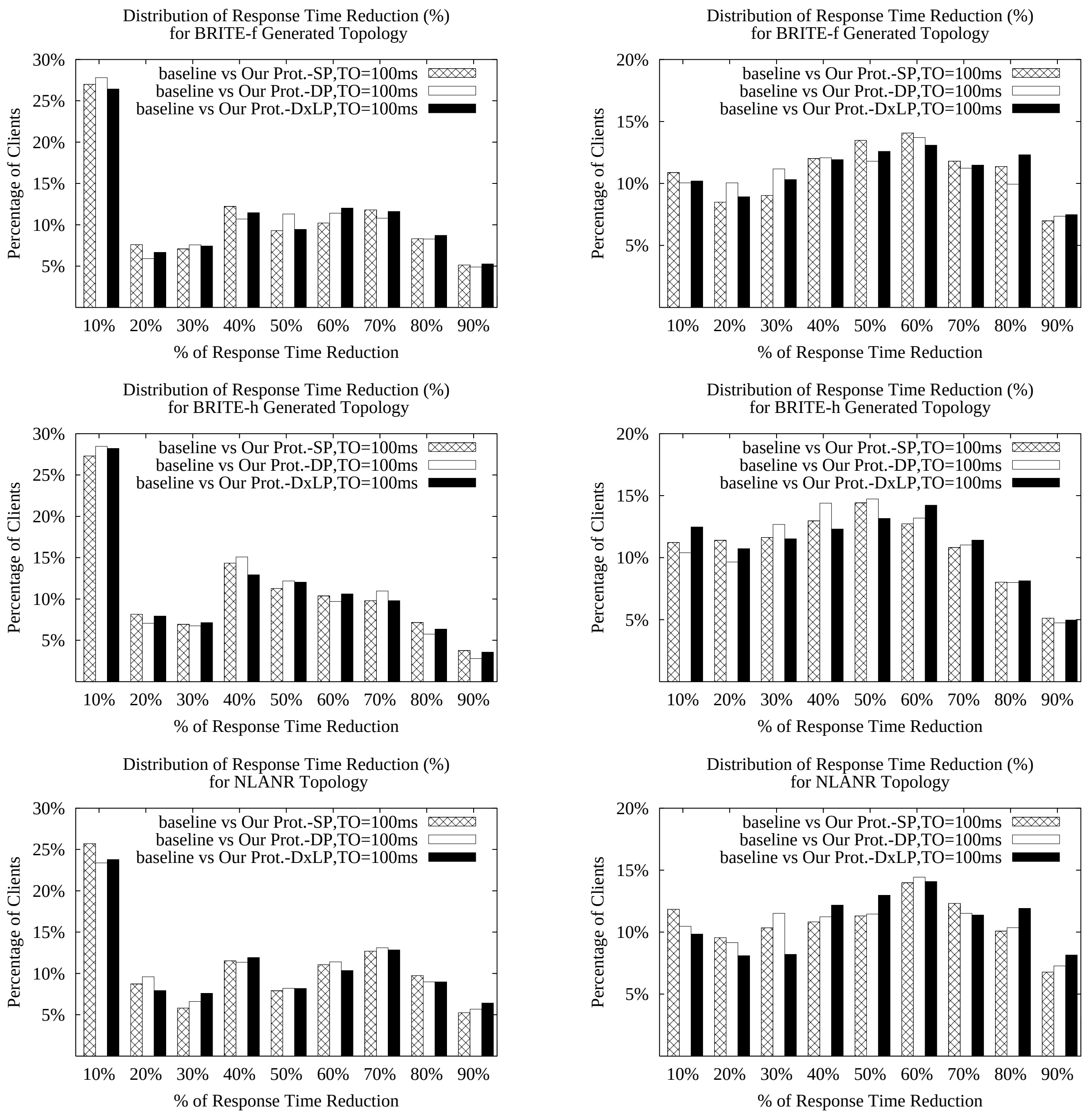


(a) Public Network
Connecting Edge Servers and Data Centers



(b) (Virtual) Private Network
Connecting Edge Servers and Data Centers

Figure 8: Browser Perceived Response Time CDF - NLANR Topology.



(a) Public Network
Connecting Edge Servers and Data Centers

(b) (Virtual) Private Network
Connecting Edge Servers and Data Centers

Figure 9: Distribution of Browser Perceived Response Time Reduction.

longer than the alternative path selected by the other policies.) Reduced advantages are observed for the BRITE-h topology (see Figure 7(a)) where, despite the relevant amount of path-diversity between clients and edge servers (see Table 1), the hierarchical organization of the network topology does not favor disjointness in between the edge servers and the back-end data centers. Also, network paths between edge servers and data centers result significantly longer than network paths between clients and edge servers, which, together with the reduced level of disjointness, additionally contributes to reduced effectiveness of the parallel invocation scheme.

The results related to the case of communication between edge servers and data centers via a (virtual) private network (see Figures 6(b), 7(b) and 8(b)) confirm the previous tendencies, with the only observation that, compared to the case of Internet based communication, this time we expect higher system responsiveness due to the more controlled network behavior at the side of the Web infrastructure (recall that for this configuration the parameter p has been set to obtain the extremely reduced packet loss rate of 1% over a path). Hence, the advantages from the parallel invocation protocol need to be evaluated for response time on the order of the reasonable value of 3/4 seconds, which is guaranteed by the parallel invocation scheme in about the 90% of the cases. Instead, even in such a controlled network scenario, the baseline protocol guarantees that response time value only in the 80% of the cases.

Another important observation from the plots is that they show significant benefits from the parallel invocation protocol even in case of no exploitation of path correlation information in the selection of the edge servers to be contacted in parallel by the client. In fact, the benefits from the correlation unaware selection scheme, namely SP, are in practice identical to those achievable with the other selection policies. This is an interesting result that confirms the feasibility of our parallel invocation protocol also in environments where it is difficult (or impossible) to infer the path correlation of the underlying network topology.

The plots in Figure 9 provide a different perspective to quantify the benefits achievable through our proposal. In these graphs we report the histograms of the percentage reduction in response time over the baseline for all the three considered network topologies and for the three edge server selection policies SP, DP and D×LP. This data visualization highlights that there is a relevant percentage of clients experiencing a remarkable reduction in the perceived response time (evaluated as $\frac{Time_{baseline} - Time_{multi\_path}}{Time_{baseline}}$) when the parallel invocation scheme is used. Specifically, in all the topologies at least the 50% of clients get response time reduction greater than (or equal to) 50%. Also, the 25% of clients get response time reduction of at least 70%. This result also highlights the ability of our protocol to increase the predictability of the end-to-end latency.

5 Related Work

Over the years, a number of solutions have been proposed based on the idea of simultaneously exploiting multiple paths among communicating parties in order to provide enhanced performance and reliability. The common base underlying these approaches is to leverage the inherent path-diversity of multi-hop networks so to reduce the likelihood to incur in link congestions or failures. Dispersity Routing [46] and IDA [56] were probably some of the first works in this area, which proposed to split the transfer of information over multiple paths to enhance dependability and performance. Simulation results and analytical studies have shown the benefits of this approach [7, 13] in the context of real-time communications. Compared to these results, we aim at leveraging existing ADN infrastructures to exploit multiple paths without requiring explicit path-diversity support from the network layer.

Related, but orthogonal to our work, are the recent studies in [32, 35, 65, 66] which are aimed at characterizing and measuring the degree of path-diversity provided by current large scale network infrastructures. The results reported in these works show that the Internet has the potential for a high level of path-diversity, especially towards multi-homed hosts [35] (as it is often the case for popular servers [32]). Additionally, selection policies for alternate paths, based on some form of topological information [34, 35, 74, 76], can help maximizing path disjointness.

Recent works close to our approach exploit path diversity and redundancy for Quality-of-Service (QoS) purposes in content delivery applications, such as parallel file downloads [15, 19, 21], cooperative Web cache sharing [37, 71], multimedia streaming [6, 40], real-time voice communication [41] and access to non-transactional Web-services [49]. Generally speaking, a promising result highlighted in [6, 19] is that existing content/service delivery (overlay) infrastructures seem to have the intrinsic potential for providing uncorrelated network paths among a client and multiple edge servers, even though these infrastructures were originally designed to minimize distance from clients to edge servers rather than for maximizing path disjointness. Compared to the above mentioned solutions, our approach has the distinguishing feature of coping with requests that trigger transactional manipulations of application data. For this scenario, apposite mechanisms to properly handle parallel invocation over multiple paths are required (i.e. the smart connection phase between edge servers and back-end data centers in our proposal) to address the additional difficulties that arise with respect to content delivery applications (e.g., non-idempotent request processing or mutual blocking due to data conflicts among transactions originated by parallel transmission of the same client request).

In [49], 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 [49] 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 if the response message is large with respect to the available bandwidth at the client. Our protocol intrinsically provides the same advantages since a single response message is returned to the client (i.e. the one from the edge server that really executes the business logic), hence not requiring large bandwidth at the client to reveal effective ([5]). Furthermore, allowing the client to explicitly choose the more

[5]In transactional Web-based applications, request messages are significantly smaller than responses [45].

responsive server, as in [49], requires periodic network probing. Our approach avoids client probing mechanisms, thus embedding within the request handling scheme all the mechanisms for autonomous selection of the currently best performing server, among the ones contacted in parallel.

Our proposal is also related to a number of results in the field of leader election (see, e.g. [1, 14, 53]). Specifically, based on the management of Connection Lists, our parallel edge server invocation protocol determines a single edge server, among the ones contacted by the client, which is responsible for processing the (distributed) transaction on the back-end data center(s). This edge server acts therefore as the leader for request processing. The innovative contribution of our protocol, compared to existing leader election protocols, is that none of them uses responsiveness while connecting to a distributed set of data centers as the election criterion.

Parallel invocation schemes are also at the basis of long established active replication techniques [59, 60]. In such a context the *replicated invocation* problem, originally analyzed in [47] and later addressed in, e.g., [26, 30, 77], shows similarities with the problems we have tackled with our proposal. Specifically, the *replicated invocation* problem arises when an actively replicated server, say S, invokes, in its turn another (replicated) server, say R. In such a scenario, the processing of the client request at the different S server replicas originates duplicate invocations towards R. This leads to inconsistencies of the state of R if the invoked application logic is not idempotent. Existing solutions to the *replicated invocation* problem (e.g., [26, 30, 47, 77]) are based on pre-filtering redundant invocation requests via an a-priori designated coordinator at the invoker side. In other words, parallelism (and dynamic optimization) does not take place along the whole chain of involved components (since the filtering scheme blocks all the outgoing requests except those from a single replica of S). Contrariwise, in our proposal parallelism and exploitation of path diversity and redundancy takes place across the whole chain of components involved in the end-to-end interaction (i.e. from the client towards the back-end data centers). This is a reflection of the fact that traditional active replication techniques are targeted to fault-tolerance and system availability, while our proposal addresses the orthogonal issue of QoS in terms of end-to-end latency.

Consistency issues arising in replicated environments have been also addressed by the work in [24] via the introduction of the X-ability framework, which defines a set of correctness criteria formalizing the obligations of a replicated service to its clients and its environment, i.e. any third-party service involved in the processing of clients' requests. Independently of whether replication is exploited for fault-tolerance or performance, an X-able service must ensure exactly-once update semantic for the state (if any) of the replicated servers and of external components (e.g. origin sites in the edge computing infrastructure). As discussed in Section 3.1.3, our protocol ensures that exactly one among the edge servers contacted in parallel by the client activates the transactional logic and propagates soft state updates

Hence the large part of bandwidth consumption at the client is related to the response message coming from the edge server.

(if any soft state is maintained) towards the other edge servers, which trivially guarantees exactly-once semantic for the update of the application state (at both the edge server and the data centers) ([6]). It is therefore relatively straightforward to show that our protocol defines an X-able service, where the replication scheme acts on a per client basis (since each client has its own set of well suited edge server replicas to be contacted) and is aimed at performance improvements via the exploitation of path-diversity.

To our knowledge, there are only two protocols addressing parallel invocation of transactional Web applications [57, 58]. Compared to the present proposal, the work in [58] addresses the limited and simple case of single back-end data center, hence not allowing parallel invocation for, e.g., e-Commerce applications involving multiple business parties, each represented by a different data center. The solution in [57] can cope with the case of multiple back-end data centers. However, differently from the present proposal, it does not address the case of variance of the communication delay between the edge server and the whole set of back-end data centers. In fact, in that solution, the edge server eventually taking care of transaction processing might be responsive towards some data centers, but non-responsive towards others. In other words, this protocol is not designed to satisfy QoS requirements specified by performance contracts. Instead, the present proposal has explicit control on the effects of variance, and adopts a self-exclusion mechanisms of contacted edge servers exhibiting high variance while connecting towards the back-end data layer. This favors low and predictable latency, thus targeting the aforementioned QoS requirements.

6 Assessments and Conclusions

Leveraging path-diversity in the context of dynamic applications executing transactions spanning multiple data centers poses a number of additional issues, that are not present in classical content delivery applications. In this paper, we have first shed lights on the additional issues that need to be addressed in order to effectively employ parallel invocation schemes in the context of applications performing distributed transaction processing. Then we have presented a novel and lightweight protocol addressing those issues through a smart approach for the management of the connection establishment phase between the edge server and the back-end data centers. Our solution does not require explicit coordination among the edge servers concurrently contacted by the client application. In fact, dynamic selection of the currently best performing path for the end-to-end interaction is based on a self-exclusion mechanism of less responsive edge serves. Our proposal is inherently scalable and does not require the underlying edge computing infrastructure to provide controlled throughput and latency among the replicas of the edge server. Finally, we provided an evaluation of the benefits from our solution, with respect to a baseline approach not exploiting redundancy and diversity, which

[6]We recall that this property is guaranteed independently of the relative speed of the chains of components simultaneously involved in the processing of a client request, as long as the occurrence of failures affecting the data centers or the edge servers are masked by orthogonal, local, fault-tolerance techniques, such as [9, 10, 22, 72].

confirmed the effectiveness of our proposal in a wide variety of system settings.

Acknowledgments

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and the ARISTOS project (PTDC/EIA-EIA/102496/2008).

References

[1] M. Aguilera, C. Gallet, H. Fauconnier, and S. Toueg. Stable leader election. In *Proc. of the 54th Conference on Distributed Computing (DISC)*, Lecture Notes in Computer Science, pages 108–122. Springer, 2001.

[2] M. Allman. A web server's view of the transport layer. *ACM Computer Communication Review*, 30(5):10–20, 2000.

[3] C. Amza, A. L. Cox, and W. Zwaenepoel. Data replication strategies for fault tolerance and availability on commodity clusters. In *Proc. of the Conference on Dependable Systems and Networks (DSN)*, pages 459–472, 2000.

[4] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. The case for resilient overlay networks. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems*, pages 152 – 157, 2001.

[5] J. Apostolopoulos and M. Trott. Path diversity for enhanced media streaming. *IEEE Communications Magazine*, 42(8):80 – 87, 2004.

[6] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On multiple description streaming with content delivery networks. In *Proc. of the 21th Conference on Computer Communications (INFOCOM)*, volume 3, pages 1736 – 1745. IEEE Computer Society Press, 2002.

[7] A. Banerjea. Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels. In *Proc. of the 2nd Conference Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 194–205. ACM Press, 1996.

[8] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns - characteristics and caching implications. In *Proc. of the 8th World Wide Web Conference (WWW)*, volume 2, pages 15–18, 1999.

[9] R. Barga, D. Lomet, S. Agrawal, and T. Baby. Persistent client-server database sessions. In *Proc. of the 7th Conference on Extending Database Technology (EDBT)*, pages 462–477. Springer, 2000.

[10] R. Barga, D. Lomet, G. Shegalov, and G. Weikum. Recovery guarantees for internet applications. *ACM Transactions on Internet Technology*, 4(3):289–328, 2004.

[11] A. Begen, Y. Altunbasakm, and O. Ergun. Multi-path selection for multiple description encoded video streaming. In *Proc. of the International Conference on Communications (ICCC)*, volume 3, pages 1583–1589, Washington, DC, USA, 2003. IEEE Computer Society.

[12] A. C. Begen, Y. Altunbasak, and O. Ergun. Fast heuristics for multi-path selection for multiple description encoded video streaming. In *Proc. of the International Conference on Multimedia and Expo (ICME)*, volume 2, pages 517–520, Washington, DC, USA, 2003. IEEE Computer Society.

[13] A. Bestavros. An adaptive information dispersal algorithm for time critical reliable communication. *Network Management and Control*, 2:423–438, 1994.

[14] J. Brunekreef, J. Katoen, R. Koymans, and S. Mauw. Design and analysis of leader election protocols in broadcast networks. *Distributed Computing*, 9(4):157–171, 1996.

[15] J. W. Byers, M. Luby, and M. Mitzenamcher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proc. of the 18th Conference on Computer Communications (INFOCOM)*, pages 275–283. IEEE Computer Society Press, 1999.

[16] N. Cardwell, S. Savage, and T. Anderson. Modeling the performance of short TCP connections. Technical Report November, Computer Science Department, Washington University, 1998.

[17] J. Challenger, A. Iyengar, and P. Dantzig. Scalable system for consistently caching dynamic web data. In *Proc. of the 18th Conference on Computer Communications (INFOCOM)*, pages 294–303. IEEE Computer Society Press, 1999.

[18] J. Chen. *New Approaches to Routing for Large-Scale Data Networks*. PhD thesis, Rice University, 1999.

[19] L. Cherkasova. Optimizing the reliable distribution of large files within CDNs. In *Proc. of the 10th International Symposium on Computers and Communications (ISCC)*, pages 692–697. IEEE Computer Society Press, 2005.

[20] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web service choreography interface (wsci) 1.0.

[21] R. Collins and J. Plank. Downloading replicated, wide-area files - a framework and empirical evaluation. In *Proc. of the 3th Symposium on Network Computing and Applications (NCA)*, pages 89–96. IEEE Computer Society Press, 2004.

[22] A. Correia Jr., J. Pereira, L. Rodrigues, N. Carvalho, R. Vilaa, R. Oliveira, and S. Guedes. Gorda: An open architecture for database replication. In *Proceedings of the 6th IEEE International Symposium on Network Computing and Applications (NCA)*. IEEE Computer Society, 2007.

[23] W. Cox, F. Cabrera, G. Copeland, T. Freund, J. Klein, T. Storey, and S. Thatte. Web services transaction (ws-transaction). Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation Inc., October 2004.

[24] S. Frolund and R. Guerraoui. X-ability: A Theory of Replication. *Distributed Computing*, 14(4):231–249, 2001.

[25] G. Gama, K. Nagaraja, R. Bianchini, R. Martin, W. Meira Jr., and T. Nguyen. State maintenance and its impact on the performability of multi-tiered internet services. In *Proc. of the 23rd Symposium on Reliable Distributed Systems (SRDS)*, pages 146–158, 2004.

[26] B. Garbinato, R. Guerraoui, and K. R. Mazouni. Implementation of the GARF Replicated Object Plateform. *Distributed Systems Engineering Journal*, 2:14–27, 1995.

[27] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 8(2), 1983.

[28] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of the 25th Conference on the Management of Data (SIGMOD)*, pages 173–182. ACM, 1996.

[29] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1991.

[30] R. Guerraoui, P. Felber, B. Garbinato, and K. Mazouni. System support for object groups. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA)*, pages 244–258, 1998.

[31] R. Guerraoui, M. Larrea, and A. Schiper. Reducing the cost for non-blocking in atomic commitment. In *Proc. of the IEEE International Conference on Distributed Computing Systems*, pages 692–697, 1996.

[32] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proc.of the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI)*, pages 13–13, Berkeley, CA, USA, 2004. USENIX Association.

[33] R. Gupta, J. R. Haritsa, and K. Ramamritham. More optimism about real-time distributed commit processing. In *Proc. of the 18th Real-Time Systems Symposium (RTSS)*, pages 123–133. IEEE Computer Society, 1997.

[34] J. Han, D. Watson, and F. Jahanian. Topology aware overlay networks. In *Proc. of the 24th Conference of Computer and Communications Societies (INFOCOM)*, pages 2554–2565. IEEE, 2005.

[35] J. Han, D. Watson, and F. Jahanian. An experimental study of internet path diversity. *IEEE Transactions on Dependable and Secure Computing*, 3(4):273–288, 2006.

[36] A. Iyengar and J. Challenger. Improving web server performance by caching dynamic data. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, pages 49–60, 1997.

[37] K. Johnson, J. Carr, M. Day, and M. Kaashoek. The measured performance of content distribution networks. In *Proc. of the 5th Workshop on Web Caching and Content Delivery*, 2000.

[38] M. B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition.* Packt Publishing, 2006.

[39] W. Li, W. Hsiung, D. Kalashnikov, and R. Sion. Issues and evaluations of caching solutions for web application acceleration. In *Proc. of the 28th VLDB Conference*, pages 1019–1030, 2002.

[40] Y. J. Liang, E. Setton, and B. Girod. Network-adaptive video communication using packet path diversity and rate-distortion optimized reference picture selection. *Journal of VLSI Signal Processing Systems*, 41(3):345–354, 2005.

[41] Y. J. Liang, E. G. Steinbach, and B. Girod. Real-time voice communication over the internet using packet path diversity. In *Proc. of the 9th ACM International Conference on Multimedia (MULTIMEDIA)*, pages 431–440, New York, NY, USA, 2001. ACM Press.

[42] F. Liu, Y. Zhao, W. Wang, and D. Makaroff. Database server workload characterization in an e-commerce environment. In *Proc. of the 12th Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 475–483, 2004.

[43] D. Llambiri, A. Totok, and V. Karamcheti. Efficiently distributing component-based applications across wide-area environments. In *Proc. of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 412–421. IEEE Computer Society Press, 2003.

[44] Z. Ma, H. Shao, and C. Shen. A new multi-path selection scheme for video streaming on overlay networks. In *Proc. of the International Conference on Communications (ICCC)*, volume 3, pages 1330–1334, Washington, DC, USA, 2004. IEEE Computer Society.

[45] B. A. Mah. An empirical model of http network traffic. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 592, Washington, DC, USA, 1997. IEEE Computer Society.

[46] N. F. Maxemchuck. *Dispersity Routing in Store and Forward Networks.* PhD thesis, University of Pennsylvania, 1975.

[47] K. Mazouni, B. Garbinato, and R. Guerraoui. Filtering duplicated invocations using symmetric proxies. In *Proc. of the 4th International Workshop on Object-Orientation in Operating Systems (IWOOOS)*, pages 118–126. IEEE Computer Society Press, 1995.

[48] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proc. of the 9th Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 346 – 353. IEEE Computer Society Press, 2001.

[49] N. C. Mendona and J. A. F. Silva. An empirical evaluation of clientside server selection policies for accessing replicated web services. In *Proc. of the 20th ACM Symposium on Applied Computing (SAC)*, pages 718–723. ACM Press, 2005.

[50] NLANR. National laboratory for applied network research. http://www.nlanr.net/Routing/rawdata.

[51] T. I. O. S. I. (OSI). *Commitment, Concurrency, and Recovery (CC) standard, ISO/IEC 9804.3 (1989) (service), ISO/IEC 9805.3 (1989) (protocol).* 1989.

[52] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Proc. of the 14th Conference on Distributed Computing (DISC)*, Lecture Notes in Computer Science, pages 315–329. Springer, 2000.

[53] D. Peleg. Time optimal leader election in general networks. *Journal of Parallel and Distributed Computing*, 8(1):96–99, 1990.

[54] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 288–301, New York, NY, USA, 1997. ACM.

[55] PlanetLab. http://www.planet-lab.org/.

[56] M. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[57] P. Romano and F. Quaglia. A path-diversity protocol for the invocation of distributed transactions over the web. In *Proc. of the IEEE International Conference on Networking and Services (ICNS)*, 2005.

[58] P. Romano, F. Quaglia, and B. Ciciani. A protocol for improved user perceived QoS in web transactional applications. In *Proc. of the 3th Symposium on Network Computing and Applications (NCA)*, pages 69–76. IEEE Computer Society Press, 2004.

[59] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computuer Surveys*, 22(4):299–319, 1990.

[60] F. B. Schneider. *Replication management using the state-machine approach*, chapter 7, pages 169–197. ACM Press/Addison-Wesley Publishing Co., 1993.

[61] M. Singhal. Update transport: A new technique for update synchronization in replicated database systems. *IEEE Transactions on Software Engineering*, 16(12):1325–1336, 1990.

[62] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using XML. In *Proc. of the 18th Symposium on Operating Systems Principles (SOSP)*, pages 160–173. ACM Press, 2001.

[63] Sun Microsystems. JSR 220: Enterprise JavaBeans^{TM}, version 3.0 - Java Persistence API, May 2006.

[64] Suresha and J. R. Haritsa. On reducing dynamic web page construction times. In *Proc. of the 6th Asia-Pacific Web Conference (APWeb)*, Lecture Notes in Computer Science, pages 722–731. Springer, 2004.

[65] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. Characterizing and measuring path diversity of internet topologies. In *Proc. of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 304–305, New York, NY, USA, 2003. ACM Press.

[66] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. In search of path diversity in isp networks. In *Proc. of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, pages 313–318, New York, NY, USA, 2003. ACM Press.

[67] Transaction Processing Performance Council. *TPC Benchmark^{TM} W, Standard Specification, Version 1.8.* Transaction Processing Perfomance Council, 2002.

[68] W. Vogels, D. Dumitriu, K. Birman, R. Gamache, M. Massa, R. Short, J. Vert, J. Barrera, and J. Gray. The design and architecture of the Microsoft cluster service - a practical approach to high availability and scalability. In *Proc. of the 28th Internation Symposium on Fault-Tolerant Computing Systems (FTCS)*, pages 422–431, 1998.

[69] L. Wang, J. N. Griffioen, K. L. Calvert, and S. Shi. Passive inference of path correlation. In *Proc. of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 36–41, 2004.

[70] J. Wenyu and H. Schulzrinne. Modeling of packet loss and delay and their effect on real-time multimedia service quality. In *Proc. of the 10th Network and Operating System Support for Digital Audio and Video (NOSSDAV).* ACM Press, 2000.

[71] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative web proxy caching. In *Proc. of the 6th Symposium on Operating Systems Principles (SOSP).* ACM Press, 1999.

[72] H. Wu and B. Kemme. Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In *Proc. of the 24th Symposium on Reliable Distributed Systems (SRDS)*, pages 95 – 108. IEEE Computer Society Press, 2005.

[73] M. Xiong, K. Ramamritham, J. Haritsa, and J. Stankovic. MIRROR: A state-conscious concurrency control protocol for replicated real-time databases. *Information Systems*, 27(4):27–297, 2002.

[74] W. Xu and J. Rexford. Miro: multi-path interdomain routing. In *Proc. of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 171–182, New York, NY, USA, 2006. ACM Press.

[75] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez. Caching strategies for data intensive web sites. In *Proc. of the 26th VLDB Conference*, pages 188–199, 2000.

[76] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *Proc. of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, volume 36, pages 159–170, New York, NY, USA, 2006. ACM Press.

[77] M.-W. Zhao, M.-L. E. Moser, and M.-P. M. Melliar-Smith. Unification of transactions and replication in three-tier architectures based on corba. *IEEE Transactions on Dependable and Secure Computing*, 2(1):20–33, 2005.

[78] Zona Research. The need for speed. Technical report, Zona Research, 1999.

Paolo Romano has a Master degree (2002) summa cum laude in Computer Engineering by the Rome University Tor Vergata. He obtained the PhD in Computer and Systems Engineering (2006) by the Rome University Sapienza. Since 2002 to 2008 he was a a lecturer at the Rome University Sapienza for graduate and undergraduate courses in the area of Distributed Systems, Capacity Planning, Computer Architecture and Object Oriented Programming. Since 2008 he is a senior researcher at INESC-ID, where he coordinates the research activities of the Distributed Systems Group in the area of Distributed Transactional Memories. In this area, he is the coordinator of several national and European research projects. His research interests include dependability of parallel and distributed systems, autonomic systems, performability modelling and evaluation and high performance computing. In these areas, he has published more than 30 papers and serves regularly as Program Committee member and reviewer for prestigious international conferences and journals (e.g. IEEE TPDS, IEEE TKDE, IEEE NCA, IEEE ICAS).
Francesco Quaglia received the Laurea degree (MS level) in Electronic Engineering in 1995 and the PhD degree in Computer Engineering in 1999 from the University of Rome "La Sapienza". From summer 1999 to summer 2000 he held an appointment as a Researcher at the Italian National Research Council (CNR). Since January 2005 he works as an Associate Professor at the School of Engineering of the University of Rome "La Sapienza", where he has previously worked as an Assistant Professor since September 2000 to December 2004. Currently, he is also an elected member of the Academic Senate at the same University. His research interests span from theoretical to practical aspects concerning distributed systems and applications, distributed protocols, middleware platforms, parallel discrete event simulation, federated simulation systems, parallel computing applications, fault-tolerant programming, transactional systems, Web-based systems and performance evaluation. In these areas, he has been an author of more that 100 technical articles. He is also the leader, or co-leader, of several national and European research projects targeting the above fields. He regularly serves as Program Committee Member for prestigious international conferences. He has also served as Program Co-Chair of PADS 2002 and 2010, as Program Co-Chair of NCA 2007, as General Chair of PADS 2008, as General Co-Chair of SIMUTools 2011, and is currently member of PADS's Steering Committee. Starting from 2003, he regularly serves as consultant expert of CNIPA (National Center for Informatics in the Public Administration).