Design and Evaluation of a Parallel Edge Server Invocation Protocol for Transactional Applications over the Web

Paolo Romano, Francesco Quaglia and Bruno Ciciani
DIS, Università di Roma "La Sapienza"

Abstract

Parallel invocation of edge servers within a Web infrastructure has been shown to provide benefits, in terms of system responsiveness, for both content delivery applications and non-transactional Web services. This is achieved thanks to the exploitation of path-diversity proper of multi-hop networks over the Internet, which typically reduces the likelihood of client perceived link congestion. In this work we address parallel invocation of geographically spread edge servers in the context of transactional Web-based applications. Actually, parallel invocation protocols are not trivial for this type of applications, since we need to deal with a set of issues not present in classical content delivery applications, such as (i) non-idempotent business logic and (ii) increase of the workload on the data centers. In this paper we propose a simple and lightweight parallel invocation protocol for distributed transactions over the Web, which addresses all those issues in a scalable manner by requiring no form of coordination among (geographically spread) edge servers. The results of a simulation study are also reported to show the advantages from our protocol in terms of user-perceived system responsiveness.

1 Target System Architecture

The architecture of our target system consists of a set of edge servers and of a set of autonomous back-end data centers, maintaining different data sets. The edge servers host the business logic for executing transactions against the data centers, which are responsible for guaranteeing the availability and consistency of the application data. End-users access the transactional logic residing at some edge server through a client application (e.g. an applet running in a browser), which interacts with the edge servers via the Internet. The processing of the client request at the edge server consists in the execution of a distributed transaction against the data centers, and in the computation of a response message (e.g. an HTML file) to be delivered to the client itself.

The execution of the distributed transaction can bee seen as composed of the following two phases:

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

Transactional Business Logic Execution. We abstract over the details of the transactional logic (e.g. SQL statements) which are obviously application dependent. Anyway, 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.

2 The Protocol

2.1 Data Center Behavior

Figure 1 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 it straightforwardly complies with, e.g., conventional DBMS technology). As stated, 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: (i) The identifier $ConnectingES$ of the edge server requesting connection establishment; (ii) The identifier $ClientID$ of the client for which $ConnectingES$ is currently asking connection establishment; (iii) 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 Connection messages tagged with the same $ClientID$ value with a list, which we refer to as *Connection List* for a given client identifier. In the following, we will use the notation $CL_{ClientID}$ to refer to this data structure. Each entry in $CL_{ClientID}$ 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.

By the pseudo-code in Figure 1, upon the receipt of a Connection message tagged with a given client identifier $ClientID$, the data center waits for incoming Connection messages with that same client identifier over a short timeout period (on the order of few tens of milliseconds). Afterwards, for all the received Connection messages, the corresponding edge server identifiers are inserted into the $CL_{ClientID}$ list, and then ConnectionACK messages are sent back to all those edge servers from which connection establishment requests have been already received within that timeout period.

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

Figure 1. Data Center Behavior.

1. $ESlist = \{ES_1, \ldots, ES_n\}$;
2. $ResultType\ Res$;
3. $RequestType\ Req$
4. $ClientIdentifier\ ClientID$;
5. set a unique value for $ClientID$;
6. set the content of Req;
7. **send** Request[$ClientID$,Req,$ESlist$] to each edge server in $ESlist$;
8. wait for a Response[Res] message from any edge server in $ESlist$;
9. return Res to the client application;

Figure 2. Client Behavior.

1. $DataCenterList\ DClist$;
2. $ResultType\ Res$;
3. $EdgeServerIdentifier\ ConnectingES$;
4. wait for a Request[$ClientID$,Req,$ESlist$] message from client;
5. set $ConnectingES = GetMyID()$;
6. determine from Req the list of data centers $DClist$ involved in the request;
7. **send** Connection[$ConnectingES$,$ClientID$,$ESlist$] to all $DC_i \in DClist$;
8. wait for ConnectionACK[$CL^i_{ClientID}$] from all $DC_i \in DClist$;
9. **if** ($ConnectingES$ is included in all the returned $CL^i_{ClientID}$ lists) AND
   ($ConnectingES$ is equal to the maximum value in the set of
   $MinEdgeServer$ values across all the returned $CL^i_{ClientID}$ lists)
10. execute the business logic associated with Req and compute Res;
11. **send** Response[Res] to client;

Figure 3. Edge Server Behavior.

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 all the edge servers contacted in parallel by the client). This depends on the current system conditions, which impact both the communication delay between the client and those edge servers, and the communication delay between each of those servers and the back-end data center. In case Connection messages with that same $ClientID$ arrive at the data center after the timeout has expired, the data center behaves as follows. It inserts the identifier of the connecting edge server into the $CL_{ClientID}$ list only in case the identifier is greater than the maximum edge server identifier already stored within that list. Independently of the insertion of that additional entry, the $CL_{ClientID}$ list is returned to the edge server currently requesting connection via the ConnectionACK message. There are three main observations we would like to bring to the reader's attention:

Observation 1. After the statement in line 4 is executed, the minimum edge server identifier, say $MinEdgeServer$, recorded within any entry of the $CL_{ClientID}$ list remains unchanged over time. Hence, given that any ConnectionACK message is ever sent out by the data center only starting from line 5, all ConnectionACK messages ever sent out for a given client identifier piggyback a connection list $CL_{ClientID}$ with the same $MinEdgeServer$ value.

Observation 2. The edge server with the maximum identifier in $ESlist$ eventually has a corresponding entry within $CL_{ClientID}$. In other words, the identifier of that edge server will eventually be inserted into the $CL_{ClientID}$ 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 $CL_{ClientID}$ list. We note that in case a Connection message from a given edge server is delivered at the data center within that interval (or even triggers the timeout interval, being it the first Connection message for a given client identifier), it means that this edge server is among those edge servers in $ESlist$ that, depending on current network conditions, more timely than others are both reached by the client request and, in turn, reach that data center. Hence this edge server is a good candidate for supporting the whole end-to-end interaction from the point of view of that data center.

2.2 Client Behavior

We show in Figure 2 the pseudo-code for the client behavior. It is quite simple since the client itself only needs to send Request messages in parallel to the set of edge servers in the list $ESlist$, tagged with the client identifier $ClientID$, and the request content Req (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). After having transmitted its request in parallel to the selected edge servers, the client waits for a reply from one of them.

For the sake of simplicity, we have illustrated the client behavior for the case of short lived Web-based interactions with application data in the back-end layer, i.e. the case of interactions completed through a single pair of request/reply messages at the client side. Extension to the case of long lived interactions with data in the back-end layer (e.g. sessions) is however immediate since the the client might keep sending requests within the long lived interaction to the only edge server that provided the reply to its first request message. As we will show, this is one edge server that revealed highly responsive during the early phase of the long lived interaction (i.e. the connection phase towards back-end data centers).

2.3 Edge Server Behavior

Figure 3 shows the pseudo-code for the edge server behavior. Upon the receipt of a client request, the edge server sends to each of the involved data centers a Connection message tagged with its identifier, the client identifier, and the list of edge servers contacted in parallel by the

client. It then waits for ConnectionACK messages from all the data centers. Each ConnectionACK message from the data center DC_i piggybacks the connection list associated by that data center with that client identifier. We have denoted as $CL^i_{ClientID}$ 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. At this end, it checks (see line 9 of the pseudo-code in Figure 3) whether: (A) Its identifier is within $CL^i_{ClientID}$ lists returned from all the contacted DC_i; and (B) Its identifier is equal to the maximum value across the set of $MinEdgeServer$ elements over all those lists. If both conditions in points A and B are verified, 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, the edge server simply terminates (with implicit disconnection from the data centers).

By **Observation 1** in Section 2.1, the minimum values maintained by $CL_{ClientID}$ lists at the back-end data centers does not vary over time. Hence, no two different edge servers will both find the condition in point B verified. Therefore, no two different edge servers will both proceed processing the distributed transaction on the back-end data centers. This ensures at-most-once semantic for the processing of the client request, and also limits 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).

By **Observation 2** in Section 2.1, there is at least one edge server (i.e. the one with the maximum identifier in $ESlist$) for which its identifier is eventually inserted within whichever $CL_{ClientID}$ list. Hence there is at least one application server for which the conditions in point A and B are eventually verified. This ensures at-least-once semantic for the processing of the client request. Hence, we avoid at all the problem of multiple updates at the back-end data centers due to non idempotent transactional logic.

By **Observation 3** in Section 2.1, if the identifier of an edge server is recorded within whichever $CL_{ClientID}$ list returned by the data centers, then it highly likely means that the edge server is a good candidate for supporting the whole end-to-end interaction towards all the back-end data centers. Our protocol captures this aspect since, for that edge server, condition in point A is verified. Among all those good candidates, the one which is allowed by our protocol to proceed with transaction processing is the only one for which condition B is verified together with condition A.

With respect to the previous point, we also note that, according to our protocol structure, an edge server quickly reached by the client request, which is also highly responsive towards a subset of data centers, but which is not responsive towards the remaining data centers, highly likely is not considered as a good candidate for the processing of the client request (since condition A highly likely does not hold for that edge server). Hence, our protocol addresses the issue of avoiding situations of large variance in the communication delay towards different data centers, which might impact the user-perceived system responsiveness.

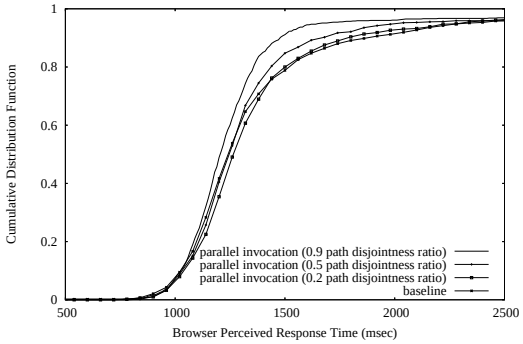

Figure 4. Response Times Distribution.

3 Related Work

Recent works that are close to our approach exploit the intrinsic diversity of large scale geographically distributed overlay infrastructures for content delivery, e.g. [5, 15, 4]. The main difference with our proposal is that we deal with transactional applications.

In [10], 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 [10] is based on the idea of contacting only the more responsive among the available servers. This solution was shown to perform better than pure parallel invocation in case the response message is large with respect to the available bandwidth at the client. Our proposal intrinsically provides the same advantages since only one response message is returned to the client (i.e. the one from the edge server that really executes the whole transactional logic), hence not requiring large bandwidth at the client to reveal effective ([1]). Furthermore, letting the client to choose the more responsive server requires periodic network probing, which is avoided in our approach.

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

To our knowledge, there are only two protocols that address parallel invocation of transactional Web applications [11, 12]. Compared to the present proposal, the work in

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

[11] 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 [12] can cope with the case of multiple back-end data centers. However, differently form the present proposal, it does not address the case of non-minimal variance for 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. The present proposal avoids this problem, and it is therefore suited for a wider range of system settings, including very large infrastructures possibly built over both private and public networks.

4 Simulation Evaluation

Simulation Model. In order to use realistic values while simulating network latencies perceived by the hosts participating in our protocol, under both normal and congested situations, we choose to employ empirical distributions of TCP round trip times (RTTs) that were found out in a number of recent works [1, 9, 2]. For space constraint we only report the plots obtained using the data in [2]. The number of paths that is expected to maximize the benefits from a parallel invocation protocol leveraging path-diversity has been shown to be on the order of two [3, 4]. Hence we focus in the simulation study on the case of two edge servers contacted in parallel by the client. Also, according to the results in [4], the two paths from an end-point to different edge servers are modeled as having a symmetrical disjointness ratio, defined as $N_{disjoint}/p$ (where $N_{disjoint}$ is the disjoint path length and p is the total path length), which ranges from 0.2 to 0.9.

Since the previous studies on both RTT distributions and network paths disjointness mostly deal with a "client oriented" perspective, we have decided not to use the related results in the simulation study for what concerns the network model in between the edge servers and the back-end data centers. Hence, we have taken the stance of simulating the case of a Web infrastructure with edge servers connected to the data centers through a communication network providing relatively low/controlled latency, e.g., a (private) dedicated WAN under the control of the ASP owning the infrastructure. Such a latency has been selected according to [7], and is on the order of 100 msecs average RTT.

For what concerns the transactional workload model, we rely on the so called "shopping workload", namely the reference transaction profile specified by TPC-W [13]. By the characterization of TPC-W performed in [8], the database page reference pattern for such a workload exhibits 96.6% of page references in read only mode, and 3.4% of page references in write mode. For what concerns the size of the data set maintained at each data center and other system settings, we again exploit the study in [8], where a global data set size of about 20 GB has been presented as a reasonable value for typical e-Commerce applications. For this data set size, the characterization of the shopping transaction profile presented in [8] shows an average number of 35 referenced pages for each interaction. Resource consumption at the data centers are explicitly simulated on the basis of the benchmarking results obtained in [8] while employing an IBM eServer xSeries 255 machine. We explicitly model also the costs for the manipulation of the Connection Lists, with the hypothesis that such a manipulation is performed according to ACID semantic (hence requiring eager disk accesses for updates), so to be able to recover from, e.g., volatile memory losses at the data center side. We consider a whole Web infrastructure consisting of three back-end data centers and twenty edge servers. Also, we consider the scenario in which the edge servers do not perform data caching, hence in our simulation study we have decided not to explicitly model resource consumption at the edge server side.

Simulation Results. We report in Figure 4 the distribution of browser perceived response times while varying the disjointness ratio between network paths from 0.2, to 0.5 and then to 0.9, and once fixed the timeout period at the data center for the receipt of connection messages from different edge servers for a same client to the value of 100 msecs (i.e. a value comparable with the expected RTT between edge servers and data centers). Each reported value is the average over a number of samples that ensures a confidence interval of 10% around the mean at the 95% confidence level. By the plot, we get that with path disjointness on the order of 0.5 and 0.9, our parallel invocation protocol achieves browser perceived response times less than 1.5 seconds in at least the 90% of the cases. This is not achieved by the baseline (i.e. no parallel invocation), which, at best, provides browser perceived response time lower than 1.5 seconds in at most the 75-80% of the cases. On the other hand, our protocol performs in practice like the baseline only for the extremely reduced path disjointness value of 0.2. Overall, compared to the baseline, our parallel invocation protocol exhibits real ability to provide higher system responsiveness as soon as there is non-minimal disjointness of network paths towards the contacted servers.

References

[1] J. Aikat, J. Kaur, F. Donelson Smith and K. Jeffay, "Variability in TCP round-trip times", *Proc. of Internet Measurement Conference*, pp.279-284, 2003.

[2] M. Allman, "A Web Server's View of the Transport Layer", *ACM Computer Communication Review*, vol.30, no.5, pp.10-20, 2000.

[3] J. Apostolopoulos and M.D. Trott, "Path diversity for enhanced media streaming", *IEEE Communications Magazine*, vol.42, no.8, pp.80-87, 2004.

[4] J. Apostolopoulos, T. Wong, W. Tan and S. Wee, "On Multiple Description Streaming with Content Delivery Networks", *Proc. of IEEE INFOCOM*, pp.23-37, 2002.

[5] R.L. Collins and J.S. Plank, "Downloading Replicated, Wide-Area Files - A Framework and Empirical Evaluation", *Proc. of IEEE NCA*, pp.89-96, 2004.

[6] K. Johnson, J. Carr, M. Day and M. Kaashoek, "The Measured Performance of Content Distribution Networks", *Proc. of Web Caching and Content Delivery Workshop*, pp.202-206, 2000.

[7] "Internet Traffi c Report", http://www.internettraffi creport.com

[8] F. Liu, Y. Zhao, W. Wang and D.J. Makaroff, "Database Server Workload Characterization in an E-Commerce Environment", *Proc. of IEEE MASCOTS*, pp.475-483, 2004.

[9] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times", *ACM Computer Communication Review*, vol.32, no.3, pp.75-88, 2002.

[10] N. C. Mendona and J. A. F. Silva,"An Empirical Evaluation of Client-side Server Selection Policies for Accessing Replicated Web Services", *Proceedings of ACM SAC*, pp.1704-1708, 2005.

[11] P. Romano, F. Quaglia and B. Ciciani, "A Protocol for Improved User Perceived QoS in Web Transactional Applications", *Proc. of IEEE NCA*, pp.69-76, 2004.

[12] P. Romano and F. Quaglia, "A Path-Diversity Protocol for the Invocation of Distributed Transactions over the Web", *Proc. of IEEE ICNS*, 2005.

[13] Transaction Processing Performance Council, "TPC Benchmark ^{TM} W", http://www.tpc.org/tpcw, 2002.

[14] D. Peleg, "Time Optimal Leader Election in General Networks" *Journal of Parallel and Distributed Computing*, vol.8, no.1, pp.96-99, 1990.

[15] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin and H. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching", *Proc. of ACM SOSP*, pp.16-31, 1999.