Increasing Cross-Domain Call Batching using Promises and Batched Control Structures

Quinton Y. Zondervan*
qyz@lcs.mit.edu
MIT Laboratory for Computer Science

In a client-server system, data integrity can be guaranteed through separation of the client and server into distinct protection domains. Such separation is achieved by restricting the communication between client and server to a well defined set of cross-domain calls. However, cross-domain calls are expensive and seriously affect the performance of the system.

In previous work, Bogle[Bog94] presents a mechanism, **batched futures**, that allows cross-domain calls to be batched, thus amortizing the overhead of the domain crossing across the batch. As calls are made by the client, they are placed in a batch. The batch is terminated and send to the server to be evaluated when a call is made to a procedure that returns a basic value such as an integer. Bogle showed that the improvement in performance is directly proportional to the average size of a batch (batching factor).

This work describes two new mechanisms that can be used to improve the batching factor: basic value promises and batched control structures.

The batched futures mechanism was limited by the fact that a batch is terminated when a call returns a basic value. However, basic value results are not always of immediate interest to the client. Basic value promises allow the batching of calls that return basic values. Instead of sending the batch to the server and waiting for the result, a promise[LS88] for the result is returned and the call is placed in the batch. The promise can later be **claimed** to obtain the actual value. Claiming a promise *will* terminate the batch.

Batched control structures (BCS) allow the batching of simple control flow statements such as if statements, while loops and for loops. Control flow in the client sometimes depends on the result of a remote call. In the case of a loop, this may require a round-trip communication on every iteration of the loop. Using special calls that place additional information in the batch, the client can batch the entire control structure. When the batch is evaluated by the server, the batched control structure is parsed, and executed. Thus, only a single domain crossing is required to evaluate the entire control structure. In addition to increasing the batch size, thus improving performance, using BCS also reduces other call overhead such as type checking and dispatching.

Both Promises and BCS have been implemented in Thor[LGJS90], a new object-oriented database system. Figure 1 shows the performance of three loops, each performing identical tasks. In all three cases the batch size is effectively the same. Each loop executed one hundred thousand iterations. The

*Quinton Zondervan is advised by Prof. Barbara Liskov and is supported by a CRFP fellowship from AT&T Bell Laboratories. Address: Room 526, 545 Technology Square, Cambridge MA 02139. Web: `http://www.pmg.lcs.mit.edu/~qyz`

first loop is a client side loop that is batched using batched futures. Because all the calls in the loop body can be batched, the loop is essentially unrolled into the batch, and then evaluated by the server. The second loop is a while BCS. Despite the fact that the batching factor is the same as in the batched futures case, the while BCS outperforms the batched futures version of the loop. This is due to the reduction of per call overhead in the BCS mechanism. The loop is parsed once by the server and executed many times. Thus the server can optimize the code before executing it, which it could not do in the batched futures case. The third loop is a foreach BCS, which calls a Thor iterator. The iterator outperforms the while BCS because it is a more concise description of the computation in this case, thus requiring fewer calls per iteration.

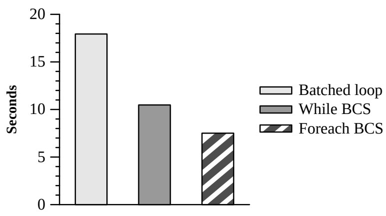

Figure 1: BCS vs batched futures.

References

[Bog94] P. Bogle. Reducing cross-domain call overhead using batched futures. Master's thesis, Massachusetts Institute of Technology, 1994.

[LGJS90] B. Liskov, R. Gruber, P. Johnson, and L. Shrira. A highly available object repository for use in a heterogeneous distributed system. In *Proceedings of the Fourth International Workshop on Persistent Object Systems Design, Implementation and Use*, pages 255–266, Martha's Vineyard, MA, September 1990.

[LS88] B. Liskov and L. Shrira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In *Proceedings of the ACM SIGPLAN '88 Conference on Programming Languages Design and Implementation*. ACM, June 1988.