A Remote Procedure Call Approach for Extreme-scale Services

Jerome Soumagne[1], Philip H. Carns[2], Dries Kimpe[3], Quincey Koziol[1], and Robert B. Ross[2]

[1]The HDF Group
[2]Argonne National Laboratory
[3]KCG

Introduction

When working at exascale, the various constraints imposed by the extreme scale of the system bring new challenges for application users and software/middleware developers. In that context, and to provide best performance, resiliency and energy efficiency, software may be provided as a service oriented approach, adjusting resource utilization to best meet facility and user requirements. These services, which can offer various capabilities, may be used on demand by a broad range of applications.

Remote procedure call (RPC) [1] is a technique that originally followed a client/server model and allowed local calls to be transparently executed on remote resources. RPC consists of serializing the local function parameters into a memory buffer and sending that buffer to a remote target that in turn deserializes the parameters and executes the corresponding function call, returning the result back to the caller. Building reusable services requires the definition of a communication model to remotely access these services and for this purpose, RPC can serve as a foundation for accessing them. We introduce the necessary building blocks to enable this ecosystem to software and middleware developers with an RPC framework called *Mercury* [2].

RPC for High-Performance Computing

RPC appears to be useful as a basis for building services for high-performance computing. However, using standard and generic RPC frameworks on a high-performance computing (HPC) system presents two main limitations: the inability to take advantage of the native high-speed transport mechanism in order to transfer data efficiently, since RPC frameworks are mainly designed on top of TCP/IP protocols, and the inability to transfer very large amounts of data, since the limit imposed by common RPC interfaces is generally on the order of a megabyte. In addition, even if no size limit is enforced, transferring large amounts of data through an RPC library is usually discouraged, mostly because of overhead from serialization and encoding, causing the data to be copied many times before reaching the remote node. Mercury is designed to address these limitations by taking advantage of native high-speed interconnects and exposing the semantics required for making nonblocking RPC as well as for supporting large data arguments, represented in figure 1.

Basis for Reusable Services

To serve as a basis for accessing and enabling reusable services in a high-performance computing environment, Mercury is designed to be both easily integrated and extended as well as providing a model that enables both high-performance and high-concurrency. It provides a network plugin mechanism that can support existing as well as future network fabrics, abstracted by a network abstraction layer. This network abstraction layer

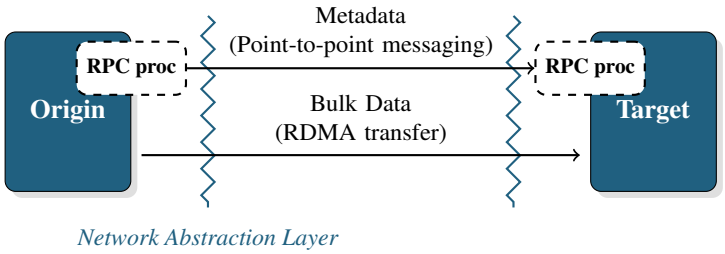

Figure 1: Architecture overview.

provides only the minimal necessary set of functionality and therefore makes it easy for developers to create a new plugin. Mercury builds upon this layer and through it defines an RPC operation as a lightweight operation, which consists of a buffer transmitted to a target where a function callback is executed. Serialization and deserialization of arguments can be either provided by Mercury or left to upper layers, which may require more specific encoding/decoding operations.

As services need to interact between each other and coordinate operations in a dynamic fashion, it is also important for Mercury processes to not be bound to a specific role, i.e. client or server. Therefore, client and server concepts are abstracted by the notion of origin and target. An origin process issues a call to a remote target process. These notions aim at simplifying the semantics and avoid any real distinction between a client and a server, since a client may also become a server in the future.

Finally to enable high-concurrency, the Mercury progress and execution model is based on a callback model, as opposed to a standard request based model. When a Mercury operation completes, a user-provided function callback is placed onto a completion queue before it gets executed. This has two advantages: first it allows upper layer services to build on top of Mercury to easily schedule operations by using for instance, a multithreaded execution model; second, it still allows definition when necessary and more convenient of shim layers that simplify common cases, based for instance on a request model to provide post/test operations.

Conclusion

Defining reusable software services at exascale is an upcoming challenge. As such Mercury will be a valuable asset and serve as a basis by providing a lightweight and modular RPC infrastructure for high-performance computing middleware, enabling both high-speed transfers and high-concurrency.

Higher-level features such as multithreaded execution, pipelining operations, or other auxiliary features such as group membership, authorization, etc, are not provided by Mercury directly, although Mercury is designed to provide the ecosystem so that these features can easily be built on top of it.

References

[1] A. D. Birrell and B. J. Nelson. Implementing Remote Procedure Calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, 1984.

[2] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross. Mercury: Enabling Remote Procedure Call for High-Performance Computing. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, Sept 2013.