A REMOTE PROCEDURE CALL APPROACH FOR EXTREME-SCALE SERVICES
Jerome Soumagne¹, Philip Carns², Dries Kimpe³, Quincey Koziol¹, and Robert Ross²
¹The HDF Group
²Argonne National Laboratory
³KCG

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 sending local function parameters to a remote target that executes the corresponding function call, returning the result back to the caller. Reusable services require a communication method in order to be remotely accessed executed on remote resources. RPC provides a network plugin mechanism that can support existing as well as future network fabrics. The network abstraction layer requires only a minimal necessary set of functionality and therefore makes it easy for developers to create a new plugin.

RPC for High-Performance Computing
Mercury is designed for high-performance computing systems. It takes advantage of native high-speed interconnects and exposes the semantics required for making nonblocking RPC as well as for supporting large data arguments. This has two advantages: first it allows upper layer services built on top of Mercury to easily schedule operations by using for instance, a multi-threaded 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.

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 by having client and server concepts abstracted by the notions of origin and target, since a server may also act as a client, and vice versa. Service Nodes (e.g., storage, visualization, etc), s₁ and s₂ are targets of s₃. Callbacks may be wrapped around pthreads, fibers, etc. This has two advantages: first it allows upper layer services built on top of Mercury to easily schedule operations by using for instance, a multi-threaded 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.

Enable High Concurrency
To enable high concurrency, the Mercury progress and execution model is based on a callback model, as opposed to a traditional request-based model. Progress

Upcoming Challenges
Defining reusable software services at exascale is a challenge. For 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, pipelined 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