Scalable Infrastructures

Dong Ahn¹, Dorian Arnold, Zhenjie Chen, David DeBonis, Bronis de Supinski¹, Lucille Frey, Joshua Goehner², Taylor Groves, Samuel Gutierrez, Mike Lang³, Greg Lee¹, Matthew Legendre¹, Nathan Hjelm³, Jim Laros⁴, Barton Miller⁵, Kevin Pedretti⁴, Whit Schonbein and Martin Schulz¹

¹Lawrence Livermore National Lab ²Roguewave Software, Inc. ³Los Alamos National Laboratory ⁴Sandia National Laboratories ⁵University of Wisconsin

Contact: Dorian Arnold, darnold@cs.unm.edu

Tree-based Overlay Networks

Tree-based Overlay Networks (TBÕNs) are hierarchically organized process networks used for scalable, high-performance data communication and aggregation. Our open-source TBÕN prototype, MRNet, has an easy-to-use C++ API and supports features like flexible network topologies, customizable data aggregation filters, multiple concurrent data streams and scalable, responsive failure recovery mechanisms. MRNet runs on UNIX/Linux clusters and supercomputers like the IBM BlueGene and Cray XT. As a result of its scalability characteristics, flexibility and wide availability, MRNet has been integrated into a variety of tools and applications and is being used at research organizations throughout the world, including Lawrence Livermore, Los Alamos and Sandia National Laboratories.

Modeling Tree Aggregation Networks

As high-performance systems continue to expand in power and size -- scalable mechanisms for communication and processing are required. Many popular frameworks utilize scalable reduction networks to fulfill the performance requirements of a large distributed system. The structures to handle this aggregation may simply consist of a single level with children connecting directly to the destination node, or it may be layered to create a large tree with varying breadth. Despite their commonplace, the techniques for modeling these Tree Aggregation Networks (TANs) are lacking. We address this need with a specialized performance model. Our performance model adheres to the simplicity of the LogP model but utilizes structural insight to provide a simple yet precise performance estimate. Additionally, our model makes no assumptions of the underlying NIC transfer mechanisms or uniformity of tree breadth, making it suitable for a wide range of environments.

Intelligent Topology Design

This work seeks to answer a fundamental question for work involving TB ÖN’s: How can we create and evolve the overlay networks that an application depends on. In other words, how well can overlay network perform on behalf of an application, given minimal specific knowledge of the application and an application developer who is not a computer systems expert.

Our approach to this problem is to use lightweight mechanisms for topology design incorporating our developed performance models. Decision processes must then be used to determine the best design principles and, in the case of performance failures, evaluate the costs and benefits of system reconfigurations. An important observation is that the definition of a performance failure is dataflow dependent; i.e. the rate of data transmission, the complexity of the aggregation operation, and the topology of the middleware infrastructure determine whether a deficiency exists -- generally defined as an under or over subscription of resources. Accordingly, corrective actions need to be considered on a per dataflow basis.

Scalable Job Startup

The lightweight infrastructure-bootstrapping infrastructure (LIBI) project targets a uniform and scalable bootstrapping process for extreme-scale-software systems. This involves launching processes on a requested set of nodes and propagating relevant initialization information to the launched processes. The LIBI API presents a consistent interface to the programmer while leveraging the native HPC services (like SLURM, ALPS or OpenRTE) when available. This enables application portability while maintaining the speed of the native services. We also developed a novel algorithm (based on our performance model) that determines an optimal bootstrapping strategy. Our algorithm can decrease bootstrap time by up to 50%. More recently we are exploring how the use of a distributed key value store may be used to facilitate further and speed up data propagation during this process.
Power Monitoring and Control

We are defining a power API framework that will expose resources for gathering power information in real-time while exploring the balance of how much information is necessary to respond to power events without causing too much additional overhead. Also, this API will expose a generic interface for throttling power usage such that the system can make better use of its power budget.

With the tools that are currently available, we are extending their capabilities in multiple dimensions varying sampling rates, exposing component level granularity, implementing embedded counters, and allowing distributed access to these readings. We expect to better quantify what is necessary to achieve the best utilization of system resources without extensive overhead cost in performance or power. Our research thus far on instrumented clusters within the Advanced Systems Technology Test Beds have allowed us a better understanding of power sensing and response requirements for future HPC systems.

Large Scale Software Testing (Open-MPI Testing Tool)

Large-scale software projects can involve many developers working at multiple locations on a variety of architectures. To enable timely identification of bugs and performance issues, regression and performance tests must be run on all of these architectures and the results must be available to all developers. The individual tests involved will also change and evolve as the software is modified.

The Open-MPI Testing Tool (MTT) has been developed to perform all these functions for the Open-MPI project. MTT is an automated suite of performance and regression tests that run at multiple locations and submits all test results to an online, freely accessible database. The user can specify any combination of MPI versions, compilers, compiler options, runtime options and tests. A single run of MTT runs all possible combinations of these options. We are reconstructing MTT into a modular framework, to simplify the code in order to facilitate easy test extensions and convenient result displays.

Large Scale Debugging

The Stack Trace Analysis Tool (STAT), a 2011 R&D 100 Award winner, is a highly scalable, lightweight debugging tool that identifies groups of processes in a parallel application that exhibit similar behavior. STAT gathers and merges stack traces from a parallel application’s processes. The tool produces 3D spatial-temporal callgraph prefix tree profiles based on series of snapshots from the application taken over time.

In 2012, STAT demonstrated successful debugging of a program running over one million MPI processes on the IBM Blue Gene/Q (BQ)-based Sequoia supercomputer. In this significant accomplishment, STAT has helped both early access users and system integrators quickly isolate a wide range of errors, including particularly perplexing issues that only manifested at extremely large scales up to 1,179,648 compute cores. The STAT team continues to investigate new research as well as convenience features that promise to make the tool even more useful and impactful.

References
