9-27-2002

Using the Emulab network testbed to evaluate the Armada I/O framework for computational grids

Ron Oldfield
Dartmouth College

David Kotz
Dartmouth College

Using the Emulab network testbed to evaluate the Armada I/O framework for computational grids

Ron Oldfield and David Kotz

Dartmouth Technical Report TR2002-433

Department of Computer Science
Dartmouth College
{raoldfi, dfk}@cs.dartmouth.edu

27th September 2002

Abstract

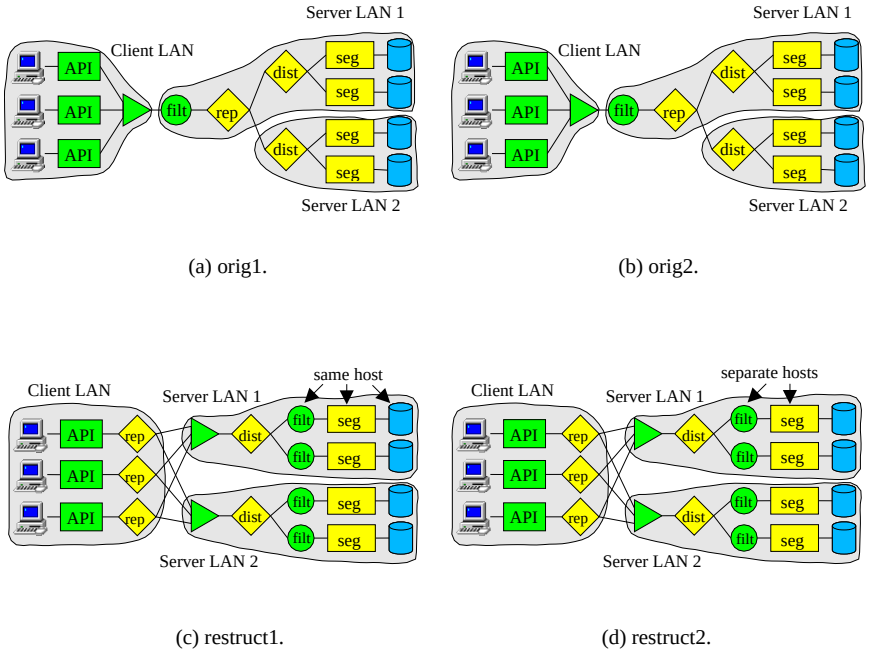
This short report describes our experiences using the Emulab network testbed at the University of Utah to test performance of the Armada framework for parallel I/O on computational grids.

Introduction

Armada [OK02a, OK02b] is a framework for building data-intensive applications for large-scale computational grids. In Armada, a graph of processing modules describe data distribution, application interfaces, and processing required of a data set before computation. The modules that make up the graph can execute on nodes near the client, nodes near the data, or intermediate network processors. Two important features of Armada include the ability to restructure the application graph to distribute computational or network load, and the effective placement of application objects to reduce network traffic–for example placing a data-reduction filter close to the data source.

There are three different environments available for testing the Armada framework: a computational grid testbed (such as the one being set up by the Global Grid Forum), network simulation, and network emulation . A grid testbed provides the most realistic environment, but does not allow direct control over network parameters; a feature that allows analysis over a wide range of conditions. Simulation provides the most flexibility with respect to control over network parameters, but producing an accurate network and computational model is often difficult; requiring a significant amount of extra work. Emulation provides a nice mix between a real network testbed and a simulated environment. Code executes on real computational resources that communicate through an emulated network topology.

This work was supported by Sandia National Laboratories under DOE contract DOE-AV6184.



(a) orig1.

(b) orig2.

(c) restruct1.

(d) restruct2.

Figure 1: Four configurations of the filtering application.

The ability to easily configure network topologies and control network parameters (such as bandwidth and latency) make the Emulab network testbed[1] ideal for examining the effectiveness and the flexibility of our approach. For example, we recently used Emulab to evaluate the performance benefit of Armada's graph restructuring algorithm on a synthetic application that filters data from two distributed data sets. For details, see [OK02b]. Figure 1 shows four configurations of the application we studied. The top two subfigures show two configurations of the original graph (labeled "orig1" and "orig2"). The lower subfigures show two configurations of the restructured graph ("restruct1" and "restruct2"). For each configuration, we ran experiments for each of seven different bandwidths, five different latencies, and three different application parameters for the filter; amounting to 420 experiments in all.

Experiment setup

We designed our network topology to have three local-area networks (LANs) connected by a single wide-area network (WAN). Each LAN is connected to the WAN through a single router, which has a link to each of the other two LANs. Figure 2 illustrates the links. Since each configuration has a similar topology, we performed all of

[1] http://www.emulab.net

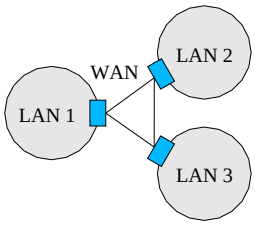


Figure 2: Topology of the network illustrating links between LANs.

our experiments on the same set of processors and used the Emulab event scheduling commands to dynamically configure the WAN network parameters between experiments.

We completed the experiments in less than five days: three days to set up the experiments, and two days to generate results. Setting up the experiments involved creating the "ns" script file to describe the topology, creating the startup scripts for each allocated processor, creating a script file to run all the experiments, and interactive testing of the application to make sure it worked correctly. There was a slight learning curve associated with the event scheduling commands, but the on-line documentation seemed to be sufficient. Most of time was spent debugging script files and the Armada code. On the fourth day, we started the full job. With the exception of a few stoppages due to bugs in the Armada code, the experiments ran without incident.

Results

Figure 3 shows timings and throughput measurements demonstrating the effect of bandwidth on the different configurations of the filtering application. In this subset of experiments, the filter removes fifty percent of the data. In (b), we also show the optimal throughputs for *orig1* (lower solid line) and the others (upper solid line).

When the WAN was slow, all configurations were limited by the WAN bandwidth. Placement of the filtering code on the server side of the WAN allowed a near-doubling in performance over *orig1*, due to the filter's halving of the WAN traffic. (*orig2* only matched the restructured graphs because it had double the WAN link bandwidth.) When the client/server WAN bandwidth was above 30 Mbps, computation associated with Java serialization and the filter code became the bottleneck. The restructured graph's distribution of the filter across four processors provided a significant performance gain over the original graph.

With the original graph, the *orig2* placement was faster than *orig1* only because its WAN links were twice as fast. When computation was the bottleneck, *orig2* and *orig1* had equivalent performance. With the restructured graph, *restruct2* was equivalent to *restruct1* at low WAN bandwidths, but was faster at high WAN bandwidths because the filter in *restruct1* shares a processor with its adjacent segment ship.

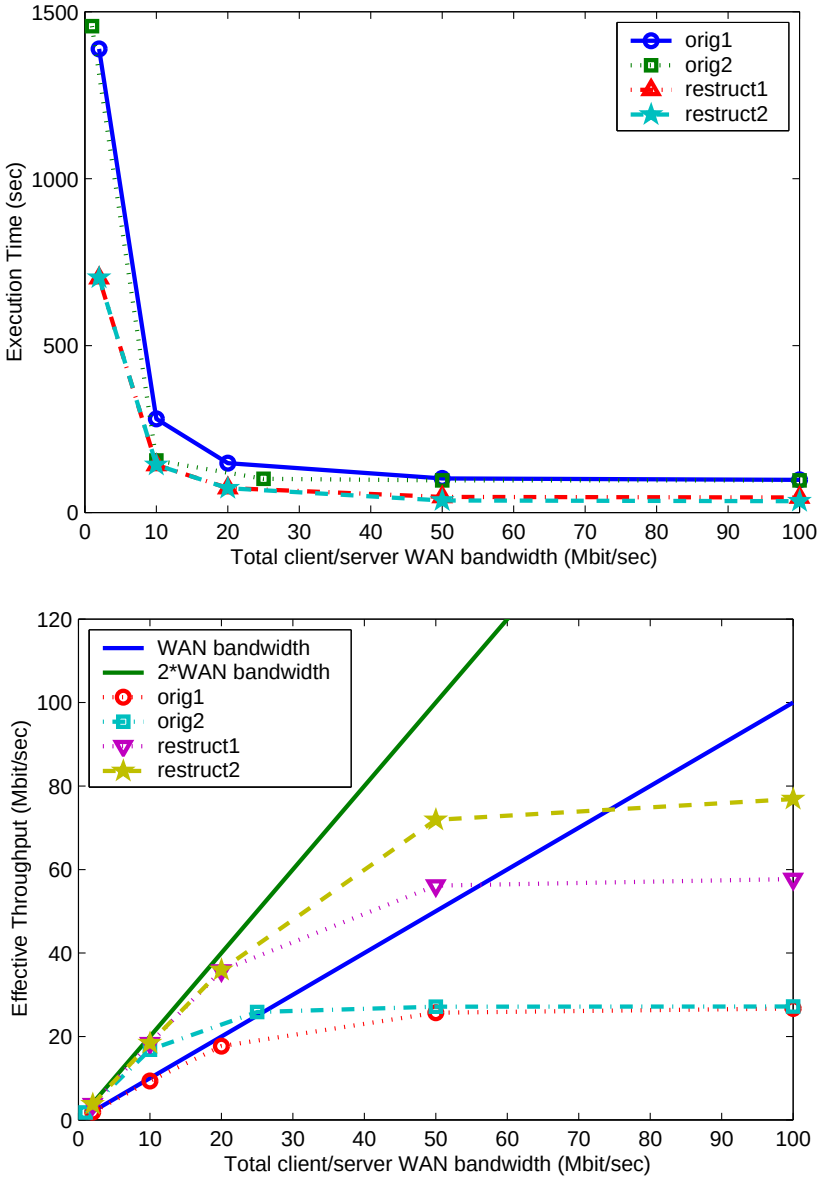


Figure 3: Time and throughput measurements of the filtering application. Each point is the mean of five independent trials.

References

[OK02a] Ron Oldfield and David Kotz. Armada: a parallel I/O framework for computational grids. *Future Generation Computing Systems (FGCS)*, 2002. Accepted for publication.

[OK02b] Ron Oldfield and David Kotz. Graph restructuring in the Armada parallel I/O framework. In preparation, April 2002.