End-to-End Data Movement Using MPI-IO Over Routed Terabits Infrastructures

Geoffroy Vallée
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831
valleegr@ornl.gov

Scott Atchley
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831
atchleyes@ornl.gov

Youngjae Kim
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831
kimy1@ornl.gov

Galen Shipman
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831
gshipman@ornl.gov

ABSTRACT

Scientific discovery is nowadays driven by large-scale simulations running on massively parallel high-performance computing (HPC) systems. These applications each generate a large amount of data, which then needs to be post-processed for example for data mining or visualization. Unfortunately, the computing platform used for post processing might be different from the one on which the data is initially generated, introducing the challenge of moving large amount of data between computing platforms. This is especially challenging when these two platforms are geographically separated since the data needs to be moved between computing facilities. This is even more critical when scientists tightly couple their domain specific applications with a post processing application.

The paper presents a solution for the data transfer between MPI applications using a dedicated wide area network (WAN) terabit infrastructure. The proposed solution is based on parallel access to data files and the Message Passing Interface (MPI) over the Common Communication Infrastructure (CCI) for the data transfer over a routed infrastructure. In the context of this research, the Energy Sciences Network (ESnet) of the U.S. Department of Energy (DOE) is targeted for the transfer of data between DOE national laboratories.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Copyright 2013 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
NDM'13 November 17, 2013, Denver CO, USA
Copyright 2013 ACM 978-1-4503-2522-6/13/11 ...$15.00.
http://dx.doi.org/10.1145/2534695.2534705

Keywords

performance measurement, networking

1. INTRODUCTION

The U.S. Department of Energy (DOE) high performance computing (HPC) facilities are spread across the country, creating many challenges such as bulk data movement and the coupling of facilities. In fact, facility coupling is becoming a critical capability since it enables the coupling of: simulation and analysis; simulation and experiments, for instance coupling light sources and neutron applications, or even in the context of high energy physics (HEP) and Fusion Energy Sciences (FES); analysis and fusion of remote sensors, for instance in the context of climate research or field campaign sensors; broader data sharing, for instance in the context of federated data centers or context delivery networks. In this context, there is a clear need for high-performance solutions for efficient data movement between DOE facilities. The goal is to achieve high-performance end-to-end data transfer between DOE facilities based on dedicated wide area networks (WAN) already deployed by DOE such as the Energy Sciences Network (ESnet).

This document presents a solution for data transfer between two MPI [3] applications coupled via a dedicated WAN. Overall, this project aims to provide an end-to-end solution for terabit data movement between remote DOE sites. The project is composed of multiple components: (i) a low-level communication layer for data movement on the Wide Area Network (WAN), based on the Common Communication Interface (CCI) [2] implementation; and (ii) a routing component for high-performance communications over the WAN in a dedicated environment. By providing these components, it is possible to provide an implementation of the Message Passing Interface (MPI) that allows developers to extend their MPI applications for the WAN or couple MPI applications, especially when transfer of large files is required. The contributions of this paper are the presentation of CCI routing, as well preliminary results using the TCP implementation of CCI, which is portable and generic.

For instance, the proposed approach can be used to easily couple MPI applications that needs to exchange data from files. As a result, the presented results are not optimized, it is for instance possible to use the same CCI prototype but RDMA over Converged Ethernet (RoCE) networks and other technologies in the future, which will provide greater performance. However, we show in this study shows the feasibility of the approach and shows results using TCP that are near end-to-end line-rate throughput when using large files.

The remainder of this paper is organized as follows, Section 2 provides a brief overview of related work; Section 3 describes CCI; Section 4 presents routed CCI; Section 6 presents a model for data transfer between MPI applications using MPI-IO and CCI; finally Section 7 concludes.

2. RELATED WORK

The transfer of files over the WAN has been an active topic of research for the past decades. Many tools, including GridFTP, BaBar Copy Program (BBCP), and bbFTP has been developed for the copy of large files over long distances. In the context of application coupling, these tools can be used to transfer output files to a remote site where the file can be fed to another application as input. GridFTP [1] is an extension to the File Transfer Protocol initially developed in the context of the Globus project [4], aiming at providing a general-purpose mechanism for secure, reliable, high-performance data movement. GridFTP has been designed to provide a framework that can be used for the construction of data-intensive tools and applications. GridFTP aims at achieving high throughput, low latency and support the connection of thousands of clients. Because GridFTP is an extension of FTP, which is not usually used for parallel computing, scientists may face challenges when extending their HPC applications. Furthermore, GridFTP is implemented using TCP and therefore can not take full benefit of part of the infrastructure that could be based on high-performance networks such as Infiniband. BBCP [5] provides a peer-to-peer secure fast copy tool and is an alternative to GridFTP. BBCP aims at supporting the transfer of large amounts of data. The data is typically fragmenting into smaller pieces that are then transfered via simultaneous streams (traditional tools such as FTP typically use a single stream). Finally, bbFTP [8] is another tool for the efficient transfer of large files. bbFTP, in contrast with GridFTP, is based on its own transfer protocol that is specialized for very large files. All these solutions have proven to be efficient for the transfer of large files over the WAN. Unfortunately, these tools are not integrated into programming languages for HPC such as MPI. This document investigates if MPI could replace these tools for the transfer of large files, providing the benefit of offering the same capability using a language scientists are already familiar with.

Another way to couple parallel applications is to use specific languages such as CORBA [7]. These solutions require the developers to learn a new language and is not necessarily available on all platforms.

The proposed work aims at providing a solution that relies on MPI to prevent developers from learning a new language and high performance like dedicated tools such as GridFTP.

3. INTRODUCTION TO CCI

The goal of the CCI project is to provide a communication interface for high-performance computing (HPC) and data centers. CCI is composed of simple and portable application programming interface (API), and is designed to be highly scalable, to enable high performance, and to be robust even in the context of faults. The CCI API can be used as a common network abstraction layer (NAL) for persistent services as well as for inter-process communication.

In the context of HPC, applications mainly rely on the Message Passing Interface (MPI) (which is a de-facto standard), and persistent services such as distributed file systems, code coupling, health monitoring, debugging, and performance monitoring. Most MPI implementations include their own NAL, whereas persistent services are mostly implemented using BSD Sockets for portability concerns. The CCI API implementation addressed these two constraints by providing an integration with an implementation of the MPI standard (i.e., Open MPI), and provides a set of implementations for different networking technologies, from Sockets (UDP), to hardware-accelerated solutions for InfiniBand and Cray's Gemini and Aries.

3.1 Key CCI Design Concepts

Events. For simplicity, CCI uses events to represent communication (e.g., send completion) as well as connection handling (e.g., incoming client connection requests). The usage of events avoids more complex programming such as the traditional Berkeley's Active Messages designed with which application developers need to use callbacks. By using events, the programming of applications is closer from the usage of Sockets, but still enables asynchronous semantics and implementations that enable high performance (zero copies implementations and so on). Finally, the event based design allows the application to perform either polling or wait; and since all events are managed via a single completion queue, CCI scalability in time is better than other solution such as BSD sockets.

Connections. Connections are used to represent the state of communication channels. By using connections, it is possible to minimize the time and space used by CCI when scaling up (in contrast to BSD sockets for instance) – no resources is allocated per connection (and therefore per peer). Connections also avoid to maintain a distributed process space such as a communicator in MPI, which greatly improve the robustness of the solution. Connections also allow applications to choose the level of service that best fits its needs and to provide fault isolation. For instance, CCI offers choices of reliability and ordering: Unreliable-Unordered (UU), Reliable-Unordered (RU), and Reliable-Ordered (RO).

Endpoints. An endpoint is a virtual instance of a device, i.e., a logical source or destination. Because of this, one endpoint can use one or more connections. For instance, if the host machine has multiple Network Interface Cards (NIC), all the available cards can be used using a connection associated with each of them to reach the same destination endpoint. Resources are allocated on a per endpoint basis (e.g., receive and send queues), which does not compromise

scalability and robustness.

3.2 Communication Modes

Based on these cores concepts, two modes of communications are provided by CCI: messages (MSG) and remote memory access (RMA). As said before, the usage of events greatly simplify the usage of the API for messages (e.g., no callbacks). For instance, an incoming message generates a receive event, which includes a pointer to the data within the CCI library's buffers. The application may access the data, copy it out if necessary, or even use the pointer to call another CCI function (e.g., the send function). Furthermore, since CCI is not using handlers, the application does not block further communication progress.

CCI also provides RMA semantics, which enables the transfer of bulk data at the application level: the application explicitly registers memory and receives a handle. The handle can then be sent to a remote peer using a message and be used to perform a RMA Read or Write operation with the remote peer. CCI RMA also supports a Fence, which can be used to ensure all preceding RMAs have completed before issuing new RMAs to that peer. RMA operations may also optionally include a remote completion message, which will be delivered to the peer after the RMA operation completes. All RMA operations require a reliable connection (ordered or unordered).

3.3 Implementation details

The CCI implementation is based on a modular architecture that enables the support of different *transports* under a single implementation of the API. In fact, a CCI transport targets a given technology such as Myrinet, InfiniBand, and BSD Sockets (UDP). The CCI implementation may provide several transports; the selection of the appropriate transport being made during the initialization of CCI. This also means that multiple transports may run side-by-side at one given type so it is possible to fully benefit from all available hardware on a given platform. In other terms, a transport is the software that enables the usage of a given NIC. In fact, CCI has the concept of *device* that represents a network interface card or host channel adapter (HCA). A device connects the host and network. Furthermore, an *endpoint* is the process' virtual instance of a device. The endpoint is the container of all the communication resources needed by the process including queues and buffers, shared send and receive buffers, etc. A single endpoint may communicate with any number of peers and provides a single completion queue regardless of the number of peers. CCI achieves better scalability on very large HPC and data center deployments since the endpoint manages buffers independent of how many peers it is communicating with.

4. CCI ROUTING

The primary goal for CCI routing is to provide end-to-end communication over heterogeneous networks across both local-area networks and wide-area networks. One usage scenario is moving data from a simulation on a leadership class system across the local-area network, over the wide-area network to another DOE facility, across its local-area network, and finally into a cluster for analysis and/or visualization. Such a scenario could transit five or more heterogeneous networks: a high-performance interconnect within the leadership class system, the local-area network, the wide-area network, the second facility's local-area network, and the cluster's high-performance interconnect. The secondary goal for CCI routing is to take advantage of the highest performing networking stack on each network. We could simply use Sockets and take advantage of the routing capabilities of the IP stack. Because of the design of Sockets precludes OS-bypass and zero-copy techniques, we need to use the non-Sockets APIs for the networks that provide those capabilities. CCI provides the ability to exploit each network's capabilities, but CCI does not provide by itself a common address space for routing. Our last goal is to provide that common address space to enable routing with minimal impact on the current CCI API and implementations. Ideally, current CCI implementations would not need any modifications to support routing.

Figure 1 shows four hypothetical organizations with one or more subnets each and all organizations are connected to the WAN. Two of the organizations, labeled AS3 and AS4, have a dedicated link separate from the public WAN. For the purposes of routing, each organization determines its routing topology and policy. We use the term Autonomous System (AS) to describe such an organization. Each AS is assigned a unique 32-bit ID. The limited number of anticipated participating organizations should allow manual assignment of AS IDs. Within an AS, each subnet is also assigned a 32-bit ID. This ID is only unique within the AS. Different organizations may use the same subnet IDs. Therefore, a specific subnet will have a globally unique combination of AS ID and subnet ID. In Figure 1, AS1 might represent a campus-wide IP network and thus it has a single subnet (SN1). It only requires a router at the border between the SN1 and the WAN. AS2 depicts multiple Ethernet broadcast domains and this organization prefers the Ethernet transport. Since the Ethernet transport requires a common Ethernet broadcast domain (EBD), this organization has to provide routers between each EBD. AS3 has three subnets: a campus wide Ethernet network (SN1), a storage-area InfiniBand network (SN2), and a leadership class compute system with a high-performance interconnect (SN3). Each subnet in AS3 has one or more routers providing connectivity to other subnets. For example, the compute system's subnet 3 is connected to subnets 1 and 2 via router 4 (R4). In addition to its WAN connectivity via router 1 (R1), AS3 also has a dedicated WAN link to AS4. This link is only valid for traffic originating or terminating at AS4 and cannot be used for forwarding to other organizations. AS4 has a campus-wide network, a storage-area network, and a couple of HPC systems.

Local Routing. Local routing is within an organization (or intra-AS). All subnets share the same AS ID. If the AS ID and the subnet ID for two endpoints are the same, the communication does not require routing at all. If the subnets have different subnet IDs, then routing is required over one or more routers within the organization, but not over the WAN.

WAN Routing. If the AS IDs differ between two endpoints, routing over the WAN is required. The routers in one orga-

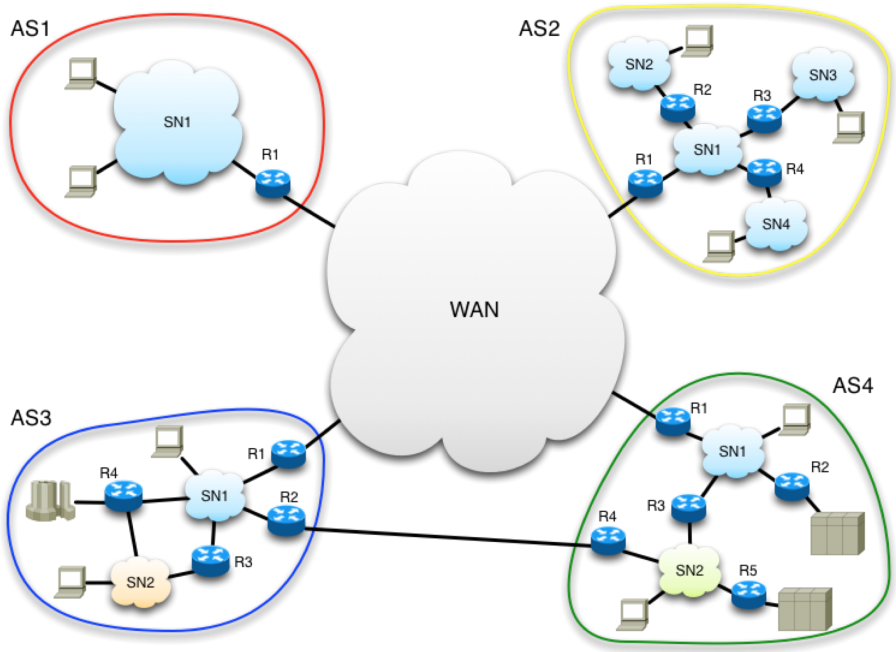

Figure 1: Multiple Organizations Each With Multiple Subnets

nization do not need, however, the complete (global) routing information for the entire path.

4.1 Route Determination

In this section, we look closer at the details of routing. Each router within an organization will need to have the same route map. The map indicates to which subnets each router connects directly as well as the path from any subnet to every other subnet within the organization. For example, a router that connects to three subnets will have three (or more) network adapters and it will have at least one CCI endpoint per subnet. Clients of the routing service will never have the map and will not be involved in the building of the routing map. Each client will have a static list of routers available within the device description in its CCI configuration file. The client will randomly choose a router.

4.1.1 Building the Complete Route Map

For this organization, we want to build routes from every subnet to every other subnet. A route will be an ordered list of one or more subnet IDs. For connections between endpoints on the same subnet, no routing is required and the route list is empty. The routing table can then be thought of as a NxN table where N is the number of subnets. The left column will be the array of originator subnets of connections and the top row will be the array of destination subnets for connections. The intersection of the row N and column M will contain the ordered list of subnet IDs from subnet N to subnet M. Altogether, AS3 has five subnets (SN1-3, WAN, and WAN to AS4). Its routing table will have five rows by five columns. We will label the rows and columns 1, 2, 3, W*, and W4 for the five subnets.

As mentioned previously, if the AS ID and subnet ID match, then we do not need to use routing. Since all routing maps are specific to an AS, the AS ID is the same for all subnets in the map. Therefore, we can identify which entries in the table that will not have any routes. In our example, the first cell is at row 1 and column 1 for subnet 1 in both cases, which does not require routing and is empty (as are 2:2, 3:3, W*:W*, and W4:W4). Also, since AS3 has two WAN links and since we do not forward through organizations, entries for W*:W4 and W4:W* are empty as well.

When subnets are directly connected via a router, the route uses a single hop. We use them to initialize the routing table. For example, router 2 connects subnet 1 and subnet 2. For the entry at row 1 and column 2 (1:2), we enter 1, 2 and at row 2 and column 1 (2:1), we enter 2,1. We continue with each router's direct connections.

Once all the single-hop routes are entered, we need to build routes between non-directly connected subnets. We will build these routes starting with the single-hop routes and combining them until we find all of the possible routes from one subnet to another. To avoid loops when computing routes, if we encounter a subnet ID a second time, we discard the route. Looking at our previous example, routing from subnet 3 to the WAN could use the following routes: 3,1,W*; 3,2,1,W*

Routing from WAN to subnet could use the reverse of these two routes. No other routes exist from subnet 3 to the WAN without incurring a loop. In addition to loop detection, we will discard routes that contain a shorter route. In the above example, the route 3,1,W* is contained within 3,2,1,W*. Since no edge (i.e. subnet) can have a zero cost to traverse, the longer route that contains a valid shorter route can never cost less than (or even equal to) the shorter route.

Note that there are no loops since all subnets 1, 2, and 3 are all directly connected to each other. And since we discard longer routes that contain valid shorter routes, the table for AS3 only has one route in each cell.

4.1.2 Choosing between multiple routes

If we encounter any loops when building the complete routing table, multiple routes will exist between some of the subnets. Likewise, if an organization has private WAN links in addition to the public WAN link, multiple routes will exist between some subnets and other organizations. The administrator will be able to choose between multiple metrics to determine which route should be used. Initially, we intend to provide bandwidth (based on link-rate, not dynamically available throughput), network capabilities (e.g. native RMA support), latency, and hop count. When choosing routes, we will use Dijkstra's Algorithm to find the shortest path in a graph.For our usage, each vertex in the graph represents one or more routers that connect two or more subnets. The edges are the subnets. For subnets that cannot be transited (i.e. the subnet can only be at the beginning or end of a route), they are represented by an edge with a vertex only at one end. When dealing with non-transiting subnets, the other vertex will represent the end node. The algorithm is a minimizing function. The goal is to find the least cost path between any two nodes. At least one of our metrics, bandwidth, needs to be converted. Ideally, the routing algorithm would choose the highest bandwidth network available. In CCI, the device information includes link-rate expressed in bits per second. To convert link-rate to a usable metric, first convert this rate to gigabits per second (Gb/s) and, second, divide a fixed, larger value by the Gb/s. For example, if the fixed value is 1 terabit per second (1 Tb/s or 1,000 Gb/s) and if the device has a link-rate of 10 Gb/s, then the metric for subnet connected to this device would be 100 (1,000/10). For a device capable of 100 Gb/s, the metric would be 10. Given a choice between a subnet with a metric of 100 (10 Gb/s) versus 10 (100 Gb/s), the algorithm will choose the lower value and pick the faster 100 Gb/s subnet. Back to our example, if the application wishes to communicate between on node on AS3's subnet 3 and a node at AS 4, it could use either the public WAN connected to router 1 or the private WAN link connected to router 2. Both routes transit subnet 3 followed by subnet 1. In our example, subnet 3 has a fast HPC interconnect with a link-rate of 64 Gb/s, subnet 2 is 10 Gb/s Ethernet as is the public WAN, and the private WAN to AS4 is 100 Gb/s. If we convert these, the bandwidth metric value for subnet 3 is 15 (rounding down), subnet 1 and the public WAN are 100 each, and the private WAN is 10. The two routes are then scored. The route through the public WAN traverses subnet 3, subnet 1, and the public WAN for a score of 215 (15 + 100 + 100). The route through the private WAN traverses subnet 3, subnet 1, and then the private WAN for a score of 125 (15 + 100 + 10). The traffic will flow over the private WAN.

To determine the cost of a route, we sum the converted link-rate metric for each subnet traversed. The route that uses subnets 1, 2, and 8 has a total cost of 300. The route that passes over subnets 1, 5, 6, and 8 has a total cost of 262 making this the preferred route. Bandwidth alone does not tell the whole story. CCI supports multiple networks including Sockets (UDP/IP and TCP/IP). Most high-performance networks provide IP or Ethernet encapsulation as well. This means that CCI can communicate over a network using its native low latency, zero-copy, and OS-bypass interface and over Sockets on top of this interface. The native interface will always perform better than the Sockets interface, but the link-rate is identical. In order to recognize this fact, the routing information will include whether the interface supports zero-copy and OS-bypass. When computing the bandwidth metric, we will convert the link-rate to the bandwidth metric and then, if the route does not use zero-copy and OS-bypass, we will double the metric (in effect dividing the link-rate by half). This will bias the routing decision to use subnets with these capabilities over subnets that do not.

5. CCI FOR THE WAN

In this section, we present CCI tuning required for the WAN, as well as a performance evaluation of CCI over a dedicated WAN.

5.1 CCI Tuning for the WAN

Transport Configuration. Users have the option of specifying CCI parameters via the configuration file. The following parameters are available: (i) bufsize: Default buffer size of the system level (for Sockets-based transports, this specifies the size of both the send and receive socket buffers); (ii) port: Specify the port used for communication; and (iii) interface: Specify the network interface to use. For the remaining of the paper, bufsize is assumed to be set to 128MB.

System Tuning. Linux systems are not by default providing an adequate system configuration for high-performance communications over the WAN. On the ANI platform, the following system parameters have been used:

```
net.core.rmem_max=134217728
net.core.wmem_max=134217728
net.ipv4.udp_rmem_min=87380
net.ipv4.udp_wmem_min=87380
net.ipv4.tcp_rmem="8192 65536 134217728"
net.ipv4.tcp_wmem="8192 65536 134217728"
net.core.netdev_max_backlog=250000
net.ipv4.tcp_timestamps=1
net.ipv4.tcp_congestion_control=htcp
```

Also, to increase performance in the context on a LAN, it can be beneficial to tune the network coalescing. For instance, between nodes on a single ANI site, e.g., ORNL, we set the network coalescing to 1 microsecond:

```
$ ethtool -C eth0 rx-usecs 1
$ ethtool -C eth0 tx-usecs 1
```

The TCP transport offers 8 tunable parameters: (i) the default value of the maximum segment size (TCP_DEFAULT_MSS); (ii) the maximum segment size (TCP_MAX_MSS), usually equal to the MTU size; (iii) the progress time

in microseconds (TCP_PROG_TIME_MS); (iv) the number of CCI receive buffers per endpoint (TCP_EP_RX_CNT); (v) the number of CCI send buffers per endpoint (TCP_EP_-TX_CNT); (vi) the number of in-flight RMA messages in the context of a single RMA operations (TCP_RMA_DEPTH); (vii) the size of RMA fragments (TCP_RMA_FRAG_SIZE); and (viii) the maximum size for any RMA fragments (TCP-_RMA_FRAG_MAX). For the remaining of this paper, the TCP transport has been tuned with TCP_RMA_DEPTH set to 1024. All other parameters are set with the default value.

5.2 Evaluation of CCI over WAN

The evaluation of CCI over WAN is based on point-to-point communications. All results are based on experimentation using the DOE-funded Advanced Networking Initiative (ANI) testbed, which is part of the Energy Sciences Network, initiating communications between Argonne National Laboratory (ANL) and Berkley National Laboratory (NERSC). This network is a 100Gb network; each node on the network having a 4x10G Myricom card; each node being setup to expose four different 10Gb ethernet cards. All tests are performed with an exclusive access to the computers and the network; and are based on the usage of a single 10Gb network card, the maximum theoretical bandwidth being therefore 10Gb. All systems are setup to use a 9000 bytes MTU. We use a ping test, which gives both the latency and the bandwidth for communications between two endpoints using small messages or RMA operations, based on the size of the messages.

Figure 2 presents the latency achieved on the ANI platform with the TCP transport (with a reliable-ordered connection) using the ping-pong test. The latency achieved with CCI is very similar to the one achieved with Netperf, i.e., about 24 milliseconds.

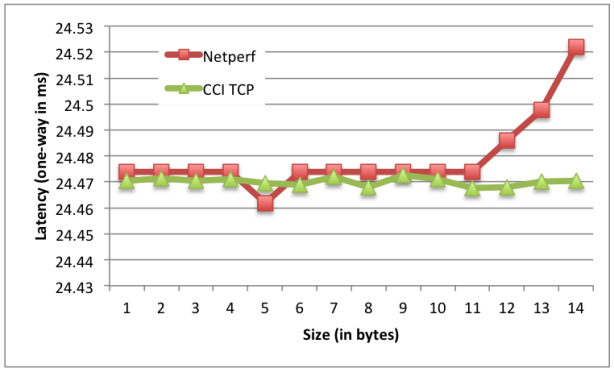

Figure 2: Ping-pong latency with the TCP transport

Figure 3 presents the RMA bandwidth with the CCI TCP transport. In the context of RMA write operations, all types of communications (RO, RU) presents a similar bandwidth since TCP is used for the implementation of the transport; only the protocol inside the transport if different for ordered and unordered connection (the transport has its own sequence number management for ordered connections). Ultimately, since the system is performing flow control and since the TCP layer is implementing the reliability protocol, it is possible to achieve higher bandwidth. This is not a streaming test; it is similar to a request/response where the request size is listed in the message size and the response is a very small message. The bandwidth tops out near 1200 MB/s at a message size of 1 GB, achieving near line-rate performance.

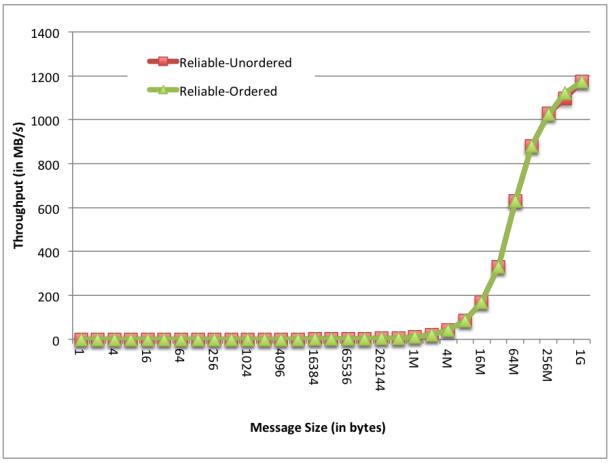

Figure 3: RMA bandwidth with the TCP transport

6. TERABIT DATA TRANSFER USING MPI-IO AND CCI

In order to demonstrate the coupling of MPI applications as well as the data transfer between the two applications, we present in this section a model based on MPI-IO and CCI for the transfer of data between two MPI applications. We also present an evaluation of the proposed model.

6.1 Data Transfer Model

This model is based on the following assumptions: (i) two sites are connected via a dedicated WAN; (ii) each site has its own separated parallel file system; and (iii) a routed communication is required for all communication between the two parallel file systems. We also assume, for simplification, that the data is read from site A and transferred to site B. Furthermore, in order to achieve high performance, parallel communications are initiated between ranks running on each site: rank i on Site A sends its data to rank i on Site B. Finally, since we assume big data files have to be transferred, ranks on site A will perform a parallel read of the file to transfer and ranks on site B will perform a parallel write to save the data into a local file once the data received. File access can be implemented using MPI-IO or the POSIX API. Figure 4 illustrates the proposed model. For the implementation of the model, CCI has been integrated into an MPI implementation: a specific Byte Transfer Layer (BTL) for Open-MPI was used; BTLs being network abstractions. It is therefore possible to use MPI for both file access, using MPI-IO, and the transfer between the two sites, using MPI communication (i.e., *MPI_Send()* and *MPI_Recv()*). The coupling of the two application is done by using *MPI_Comm_connect()* and *MPI_Comm_accept()*. Ultimately, this makes possible to couple two MPI applications using MPI, which prevent application developers from having to use specific code coupling methods and languages.

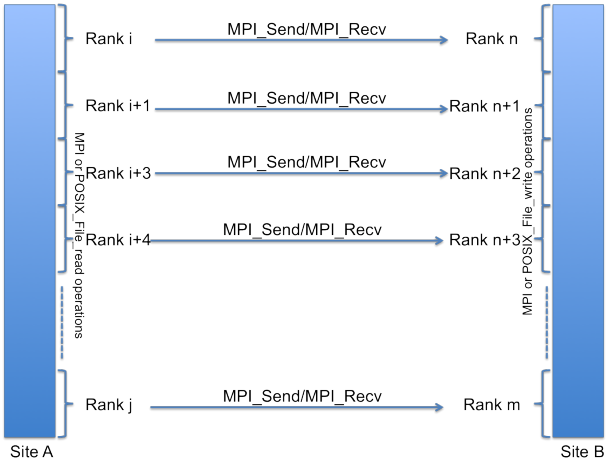

Figure 4: Data Transfer Model Using MPI-IO

6.2 Evaluation

We performed a set of preliminary evaluations using the dedicated WAN ESnet network between two hosts, one at Argonne National Laboratory (Illinois, USA) and the other at Berkley National Laboratory (California, USA). Table 1 summarizes the configuration of the nodes. The disk throughput tests were performed using the *dd* system tool.

To evaluate our work, we developed a synthetic benchmark to mimic the configuration described in Section 6.1: 2 MPI applications are running on two different sites that are connected to a WAN. The goal is to transmit a data file from site A to site B by extending the MPI application. Furthermore, since we assume we are running across a WAN, we cannot assume there is a distributed or parallel file system shared by the two sites. Site A is assumed to be nersc-diskpt-1 and Site B star-mempt-2. We first compared the transmission of a 1 gigabyte file using the default TCP BTL from Open-MPI and the CCI BTL. We also compared the results when accessing the file using MPI-IO and the POSIX API.

Figure 5 presents the results on the ESnet. The bandwidth is calculated starting a timer when all ranks start to read the file on Site A. Then each rank sends its file chunk to its corresponding rank on Site B, which writes the file upon reception. After reception, all ranks on Site B enter into a barrier to ensure the file is successfully written on the disk. Rank 0 of Site B then send a message back to Site A to notify completion of the operation. The time is stopped on Site A upon reception of this message and the throughput calculated. CCI always outperform the TCP BTL mainly because the CCI BTL enables a RMA operation for the transfer of the file, whereas the TCP BTL only uses small messages. The results also show that MPI-IO is a limiting factor when using the CCI BTL since using the POSIX API to read and write the file allows us to achieve about 400MB/s with POSIX and only about 100MB/s using MPI-IO.

Based on these results, we then optimized the benchmark to overlap file access and MPI send/receive operations. For this, we extended our benchmark based on the POSIX API: as previously each rank is assigned a chunk of the file, but the chunk is now split up in blocks of predefined size. For

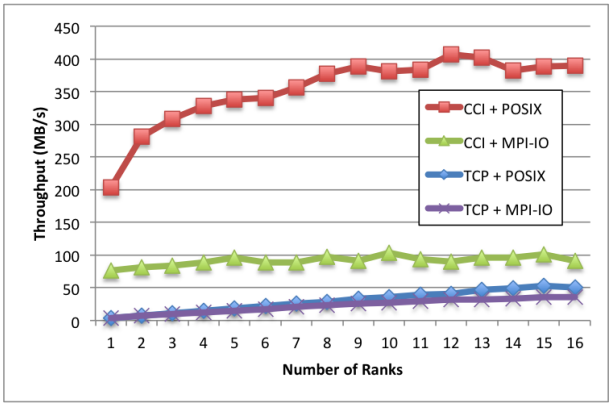

Figure 5: Data Throughput Across the WAN

instance, a 100MB file with 10 ranks and a block size of 1MB results in each rank sending 10 blocks of 1MB. For the transfer, all ranks on Site B post a non-blocking receive at the beginning of the test, before Site A starts to read the file. Ranks on Site A use non-blocking send and read/send the blocks as quickly as possible. Ranks on Site B track all the receives so we can determine overall completion.

We perform two different tests using a 1GB and a 10 GB file. For each file size, two different block size (1 and 10 MB) are used. Table 2 shows the results using 16 ranks on each site. With a 10GB file and a block size of 1MB, it is possible to achieve more than 1,000 MB/s.

7. CONCLUSION

This document presents a solution for end-to-end data movement over routed terabit infrastructures. The proposed solution is based on MPI for the coupling of MPI applications, simplifying the effort for coupling the applications; MPI-IO or the POSIX API being used for accessing the files and MPI communications for the actual data transfer between the sites. In that context, the Common Communication Interface prototype has been integrated into an MPI implementation (Open-MPI) and a proof-of-concept benchmark

Name	nersc-diskpt-1	star-mempt-2
Location	Berkley National Laboratory	Argonne National Laboratory
Number of cores	12	16
Network	10Gb Ethernet	10Gb Ethernet
Disk Write Throughput	1.7 GB/s	84.1 MB/s
Disk Read Throughput	6.2 GB/s	3.0 GB/s

Table 1: Hardware Configuration of the Testbed

Block Size	1 GB File (Thoughput in MB/s)	10 GB File (Thoughput in MB/s)
1MB	488.17	1012.42
10 MB	481.71	963.52

Table 2: Throughput Overlapping File Accesses and MPI Communications

developed to validate the proposed approach.

Ultimately the contributions of this work are: (i) the optimization of CCI for the WAN; (ii) the routed CCI; (iii) the development of a proof-of-concept benchmark; and (iv) a preliminary evaluation based on TCP, comparing CCI over TCP with the TCP substrate from Open-MPI, as well as different methods to perform file I/O.

The results show that it is possible to achieve near line-rate performance relying only on TCP and MPI for a end-to-end file transfer. This shows the benefit in this context of CCI over existing communication substrates such as the TCP BTL from Open-MPI. As a result, scientists can couple MPI applications or extend existing MPI application by using MPI, greatly decreasing the complexity of such tasks and avoiding requiring the scientists to interface their application with extra tools and/or learn new programming language. This study also shows the importance of method to access the files and its impact on the overall performance. For instance, we show that the current implementation of MPI-IO in Open-MPI introduces a very high-overhead and, as a result, it might be preferable to use other methods to read and write files (in the context of this study, the standard POSIX API provides better performance). In fact, other parts of the project are focusing on this approach, including I/O optimization solutions with layout-awareness on end-system hosts for bulk data movement [6].

CCI is designed to be extensible and therefore it is possible to support more advanced networking solutions such as RoCE or any future networking solution. We anticipate that the usage of such technologies would drastically improve overall performance and for instance achieve near line-rate performance even for smaller files.

Acknowledgment

We would like to acknowledge the individuals from The Innovative Computing Laboratory at the University of Tennessee who developed the Open-MPI Byte Transfer Layer for the Common Communication Interface prototype.

This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy and performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 and used resources of the Center for Computational Sciences at Oak Ridge National Laboratory.

This research used resources of the ESnet Testbed, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

8. REFERENCES

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 54–, Washington, DC, USA, 2005. IEEE Computer Society.

[2] S. Atchley, D. Dillow, G. M. Shipman, P. Geoffray, J. M. Squyres, G. Bosilca, and R. Minnich. The Common Communication Interface (CCI). In *Hot Interconnects*, pages 51–60. IEEE, 2011.

[3] T. M. Forum. MPI: A Message Passing Interface, 1993.

[4] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[5] A. Hanushevsky, A. Trunov, and L. Cottrell. Peer-to-Peer Computing for Secure High Performance Data Copying. In *In Proc. of the 2001 Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2001), Beijng*, 2001.

[6] Y. Kim, S. Atchley, G. R. Vallée, and G. M. Shipman. Layout-Aware I/O Scheduling for Terabits Data Movement. In *To Appear in Proceedings of ithe Workshop on Distributed Storage Systems and Coding for BigData (DSSCB 2013)*, Oct. 6-9, 2013. Held in conjunction with the 2013 IEEE International Conference on Big Data (IEEE BigData 2013).

[7] C. Pérez, T. Priol, and A. Ribes. A Parallel CORBA Component Model for Numerical Code Coupling. In M. Parashar, editor, *GRID*, volume 2536 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2002.

[8] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi. Middleware Support for RDMA-based Data Transfer in Cloud Computing. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 1095–1103, Washington, DC, USA, 2012. IEEE Computer Society.