

# The Composite Endpoint Protocol (CEP): Scalable Endpoints for Terabit Flows

Eric Weigle and Andrew A. Chien\*  
{eweigle, achien}@ucsd.edu  
Computer Science and Engineering  
and Center for Networked Systems  
University of California, San Diego

## Abstract

We introduce the Composite Endpoint Protocol (CEP) which efficiently composes a set of transmission elements to support high speed flows which exceed the capabilities of a single computer. CEP's unique capabilities include: (1) allowing multiple processes (a composite endpoint) to take part in a single logical connection, (2) providing a simple, flexible interface to describe data layouts and composite endpoint communication to user programs, (3) providing efficient transfer scheduling which coordinates heterogeneous nodes to achieve good composite performance, and (4) a scalable architecture which supports large numbers of participants in a composite endpoint.

We describe the design of CEP, an initial implementation, and an empirical evaluation exhibiting the above capabilities. We have achieved speeds over 32Gbps using commodity cluster hardware with linear scalability, performance 7× naïve approaches, and low overhead.

**Keywords:** Composite Endpoint Protocol, CEP, cluster networking, group communication

## 1. Introduction

Continued exponential increase in optical transmission speeds faster than Moore's law raises challenges in transmitting local and wide-area network flows ranging from 10 gigabit to terabit speeds. This situation is markedly different from the formative period of the

Internet where bandwidth was a scarce commodity. It is unsurprising that existing approaches [1, 7, 9, 15, 17] designed in that environment do not fully address the problems encountered in such a network.

One use for high speed WAN links is to carry millions of small flows. However, there are many scientists and other users clamoring for higher speed data transport across the Internet. If high speed, 40 gigabit trunks (scaling to terabits) are to be utilized for large flows of 100s of Gbps, it is a challenging technical problem to terminate such high speed flows in computer systems that can reasonably absorb only 1-10 Gbps today. Given the widely acknowledged gap in the scaling rates of optical transmission and Moore's law [18], this disparity is likely to continue growing.

Similarly, the rapid increase in commodity microprocessor performance has produced an explosion in the power and use of commodity clusters for computational science and even supercomputing [5, 23]. Such clusters are attractive because they are low-cost, scalable, and can be incrementally extended in computing, memory, and storage capacity by simply adding nodes. The advent of inexpensive gigabit networking in such cluster nodes makes them an attractive building block for *composite endpoints* capable of transmitting or receiving flows of 100's of gigabits per second. Our approach for the *composite endpoint protocol* (CEP) is based on the use of clusters as scalable composite endpoints for such high speed flows.

There are two extant examples of distributed systems in which a set of nodes cooperate to form a larger logical flow. First, peer-to-peer networks which use multiple servers for a single file download synthesize an N-to-1 communication pattern [3, 11, 19]. However, this approach is limited, requiring data pre-encoded with erasure codes [6, 7] and distribution of the files in advance, limiting them to read-only sharing of large files.

---

\*Supported in part by the National Science Foundation under awards NSF Cooperative Agreement ANI-0225642 (OptI-puter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from the UCSD Center for Networked Systems, BigBangwidth, and Fujitsu is also gratefully acknowledged.

Second, researchers in the high performance computing community have use striping to higher performance in GridFTP [1] or messaging systems MPI [2,15,20] allow a set of processes to participate in constructing a single large flow. These systems have significant limitations such as (1) requiring uniform or homogeneous hosts, (2) not addressing adaptive scheduling and transport over the wide area, instead they are only efficient local to a parallel machine, (3) generally being constrained to a single cluster, (4) having limited fault tolerance, and (5) being hard to use and configure.

Our objective is to support a general, flexible, N-to-M communication where compute nodes can be combined to achieve high speed flows. The composite endpoint protocol (CEP) addresses this problem. Specific contributions include:

- definition of the N-to-M communication problem, and formulation as a general approach to use scalable endpoints to support high speed network flows,
- definition of the CEP API which supports easy expression of both send-side and receive-side data layouts with overlaps to express where choice flexibility exists,
- an online scheduler and framework which produces efficient and high-performance transfer schedules, exploiting heterogeneous hardware capabilities, varied network performance, and arbitrary data layouts,
- an implementation of CEP which has proven itself useful in real-world situations,
- evaluation of a CEP implementation which demonstrates that it achieves efficient composition, with relatively small overhead and transparent incorporation of heterogeneity, composite endpoint performance is above 75% of the maximum hardware capacity,
- evaluation of CEP which demonstrates that in cases of constrained data access – common in parallel filesystem or SAN storage environments – CEP’s flexibility enables it 7× higher performance than uniform striping.

The CEP effort is part of the OptIPuter [25] project, which explores the use of configurable optical networks to provide high bandwidth and tighter coupling amongst computational and storage resource. CEP and GTP (see Section 6) provide the application-level connectivity required in this environment. Issues of network configuration, path or resource discovery, naming, etc., are provided by other tools such as the DVC [21], LambdaRouter [26], and so forth.

The remainder of the paper is organized as follows: Section 2 defines the N-to-M problem and defines solution criteria. Section 3 discusses the design and algorithms of CEP. Section 4 describes an implementation of the ideas in CEP. In Section 5, we explore the performance of a CEP implementation empirically. A discussion of the results and related work follows in Section 6 and we conclude with a brief summary in Section 7.

## 2. The N-to-M Problem

Our objective is to support a general, flexible, N-to-M communication where nodes can be combined to achieve high speed flows. To support general-purpose communication on the widest range of hardware resources, we make no assumptions of special encoding or hardware homogeneity. More formally, the problem is given an arbitrary set of  $n$  sender nodes and sender data distribution (with overlaps) and an arbitrary set of  $m$  receiver nodes and receiver data distribution (with overlaps), transfer the data as quickly as possible. Similarly, we make no assumptions about pairwise network properties. Degrees of freedom include: which sender node should transfer a segment of data, which node should receive it, and what order should data elements be transmitted. An example of this situation is depicted in Figure 1.

Nodes  $S_1$  to  $S_4$  (the Senders) have access to overlapping subsets of a data collection. The senders map their data to the logical linear data space. Nodes  $R_1$  to  $R_6$  (the Receivers) want arbitrary disjoint subsets of the linear data space. Senders and receivers may each have different capabilities (indicated by the thickness of each circle). In some case, not all data in the linear data space is desired by a receiver. The objective is to move the data from the sender to receivers as rapidly and robustly as possible in conformance with the map.

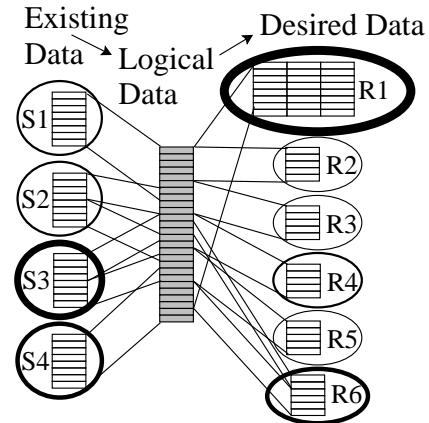


Figure 1. Sample Problem

Because the objective is to aggregate the communication capabilities of multiple nodes, the data originates from multiple sender nodes and sinks to multiple data receivers. The association of senders and data is typically not disjoint – sending nodes have access to overlapping data regions, often due to underlying shared storage systems or dedicated cluster interconnects which provide the ability to share data efficiently. The performance between distinct pairs of sender and receiver nodes can vary widely due to node hardware performance (computer, interface), different OS protocol stacks and version, misconfiguration, and other performance anomalies. In this setting, the objective is to deliver robust, high performance that approaches the underlying hardware capabilities.

A solution to the problem must include the following two parts: an API for describing the N-to-M transfer on both the sender and receiver nodes, and a scheduler which decides which node sends and receives each segment of data and when it does so.

An API determines how the problem is represented, and includes constraints on which nodes can send which data and which receivers must ultimately receive which data. The scheduler decides the sequence of data movement so as to optimize the performance of the transfer and must account for the constraints expressed in the transfer, as well as the heterogeneity in performance from the nodes, node pairs, etc. Such heterogeneity turns out to be an important problem because often nodes in “homogeneous” clusters are not truly identical. This can be the case due to differences in software configuration, network port configuration, upgrades, hardware additions and even hardware failures. Scheduling which ensures robust, high performance fundamentally increases the usability of these systems.

A good solution has the following properties:

- **Easy to Use:** No detailed hand tuning is required. A simple specification of the accessibility is all that should be required to achieve the system’s performance.
- **Efficient Composition:** The solution should allow node capacity to be efficiently aggregated to achieve ultra-high performance logical flows.
- **Tolerance of Heterogeneous Performance:** Aggregation should include performance heterogeneity, seamlessly adding capacity and delivering it to the logical flow.
- **High Performance:** System delivers all of the hardware performance available to the application in nearly all circumstances.

In the following Sections, we describe the design and evaluation of the *composite endpoint protocol*, a solution to the N-to-M high performance communication problem.

### 3. The Composite Endpoint Protocol

We describe the design of the Composite Endpoint Protocol which is our solution to the N-to-M problem. Ease of use involves a simple specification of transfers through the CEP API, and a simple interface for control of transfers. The transfer scheduler accepts that information, processes it, and generates a schedule which achieves the goals of high performance, exploiting heterogeneity, and efficient resource use.

A life cycle of a CEP transfer is as follows:

```

User provides high-level transfer data to CEP.
Loop {
    CEP generates/adjusts a transfer schedule.
    Readers/writers implement the schedule.
} Transfer completes

```

At any time in this life cycle, a user may request status information or terminate the transfer. This use model reflects our primary interest in large bulk transfers which surfaces in our metric of aggregate bandwidth rather than latency and fairness – although we will also provide reasonable performance in these other dimensions.

#### 3.1. Describing a Transfer: the CEP API

A key element of composite communication is the specification of a logical flow across a set of contributing nodes using the CEP API. To describe such a flow the user must provide a list of senders, receivers, the data each sender can access, and the data each receiver desires. The CEP programming interface expresses this structure using an extension to the idea of sequence numbers. The data any sender or receiver is responsible for is simply specified as the union of a set of disjoint numerical ranges. This notion is illustrated in Figure 1 where the “logical” segment corresponds to the sequence number space. This information can be provided as two lists of tuples, one for the senders (writer) and one for the receivers (readers) as shown in Figure 2. At a high level, this data gives the type of node, its location, and a mapping between local and global data name spaces.

This information is presented to the CEP implementation in a variety of ways defined by the CEP API (Table 1), and discussed below. The `dbInfo *dbs`

```

id "My Data File"
# Node Type  Host:Port          Data URI              Offset in file        Start-End Range
master       localhost:55555     #(Local namespace)   (Global namespace)
writer      node1:55553        "file:///data/half1" 0                      0-1023
writer      node2:55554        "file:///data/half1" 1024                   1024-2047
writer      192.168.3.4:55555 "file:///data/half2" 0                      2048-3071
writer      192.168.3.5:55555 "file:///data/half2" 1024                   3072-4095
reader      node5:55557        "file:///tmp/copy1"  0                      0-2047
reader      node7:55559        "file:///tmp/copy1"  2048                   2048-4095
reader      node6:55558        "file:///tmp/copy2"  0                      0-4095

```

**Figure 2. Sample Transfer Specification File**

structure contains fields to specify the columns in Figure 2.

The example shows four senders each with access to a disjoint segment of a source file, perhaps via a high-bandwidth shared filesystem, and three receivers interested in different sets of that data. Note that other transfers may involve different number of senders, receivers, different data overlap, and location. CEP calculates an appropriate schedule based upon this information.

While a user could call the CEP API directly or write such transfer specification files, we expect N-to-M transfers will be generated automatically by applications such as MPI programs or a parallel filesystem which knows the location and replication structure of a distributed logical object and wants to move it quickly. Such higher-level interfaces vary between applications; we are implementing common patterns including a file transfer, shared-memory-like, and sockets-like interface for inclusion in the DVC [21]. The simplified file transfer implementation is already complete. However, a detailed discussion of the application specific APIs is beyond the scope of the paper.

Each transfer is distinct and has its own unique identifier. This is in effect creating a distributed namespace analogous to a distributed file or memory system; but optimized for a specific data transfer.

### 3.2. Transfer Scheduling

CEP takes the user transfer requirements and generates an appropriate schedule. While certain formulations of the problem are known to be NP-Complete [10] by allowing transfers to be arbitrarily subdivided we avoid the issue.

First, CEP creates a table representing the user requirements. This can be thought of as a directed graph whose nodes are the senders, receivers, and simple data ranges, with links between those who provide or require each segment (senders→data blocks→receivers).

A transfer schedule is weighting of links such that each node has a nonzero-weighted link to each data segment they require, and each data segment is likewise provided by a receiver via a nonzero-weighted segment. Minimizing the transfer time is equivalent to producing a globally balanced weighting. While we have multiple scheduling algorithms appropriate for different circumstances, we present only two here: a greedy algorithm which is the one used in our implementation, and a more theoretically optimal algorithm.

**Greedy Algorithm** The greedy algorithm looks at the receivers and calculates the demand for each data segment in the system. It then looks at every sender and allocates their outgoing weights proportional to the demand for each data segment they provide. Lastly, senders allocate incoming bandwidth proportional to their individual demands and subdivide their requests amongst the senders proportional to their weighted speed. Feedback on dynamic transfer speed over the network can be used to further bias each receiver toward appropriate senders.

This mechanism is provably optimal when the transfer is strictly sender or receiver limited. When some nodes are sender limited and others are receiver limited, those bottlenecks can be minimized by non-proportional allocation—producing a better schedule. So, in the (expected) average case this is nearly optimal but in the worst case the schedule produced can be off by a factor of  $k$  where  $k$  is the amount of replication in the network.

**Linear Programming Algorithm** A more optimal solution employs a computationally expensive linear programming algorithm. We create the directed graph as with the greedy algorithm, but rather than just dividing out global sums of supply/demand proportionally we solve for them as variables in a system of linear inequalities. The node speeds and data accessibility become the constraints, and we apply linear programming methods to solve for the maximum. In the static

| Call   | Definition   |
|--|--|
| <code>void schedule(dbsInfo *dbs);</code>        | Schedule one part of a transfer (one row of config file) |
| <code>bool transfer(void);</code>                | Initiate or continue overall transfer                    |
| <code>unsigned long long data_sent(void);</code> | Provides status information                              |

**Table 1. The CEP API Function Calls**

case, this gives within a factor of  $1 + \epsilon$  to the optimal result, for an  $\epsilon$  inversely proportional to the total size of the transfer (because we can't subdivide the transfer of a byte). This latter mechanism has its own tradeoffs. In the Section 7 we give possible solutions.

Both scheduling algorithms seek to maximize performance (minimizing global transfer completion time) which in turn maximizes system efficiency. Both scheduling algorithms handle heterogeneity transparently, reflecting it in the allocations of transfers. To maximize per-node utilization we note that all weights are soft limits; if there are underutilized network links the bandwidth is absorbed by the remaining flows. Both can generate schedules to tolerate sender faults when there is a surviving replica of the desired data.

In general, a user will achieve best performance by specifying the least restrictive data access constraints on the senders and receivers. In that case, the CEP system will optimize globally for best performance. However, if desired, the user can indirectly control the amount of work assigned to a node by scaling the amount of data which it holds/requires. If a transfer is specified with no overlaps (a scatter/gather mechanism), the scheduler has no freedom of choice.

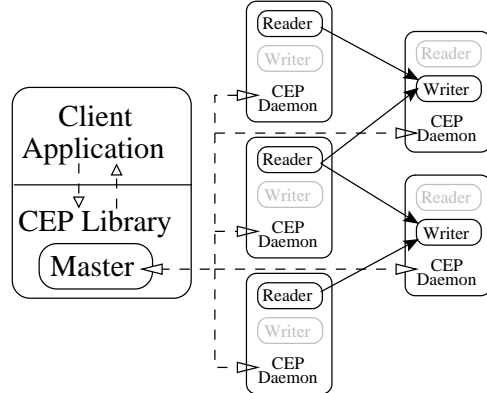
## 4. Implementation

Our implementation follows the design in Section 3, implementing the CEP API and greedy scheduling algorithm. We present information on our implementation and how a programmer or other user might integrate it with other tools. This document describes the initial proof-of-concept implementation. A revised version is in development that shows promising improvements, including higher performance, lower overhead, and functionality with multiple transport protocols.

### 4.1. Global Structure

We use a centralized, master node to implement the scheduling algorithm described above; it holds the database of constraints and uses a message-based protocol over TCP to control daemon processes which do the actual transfer. The advantage of this approach is simplicity, and we achieve acceptable scalability. In future versions, we may move to a more decentralized

model. Figure 3 shows the relationships graphically. Metadata communication is shown with a dotted line, data communication via a solid line.



**Figure 3. Overview of Global Structure**

The client application links to a library which provides the CEP API, database code, scheduler, and distributed control mechanisms. The control/data flow are as described in Section 3, with the addition of steps to communicate transfer commands from the master to the readers/writers, and retrieve status information. In other words, the metadata is centralized and processed serially, the data is distributed and processed in parallel.

We use a select-loop-driven event model based on the assumption that a single server may concurrently serve many different parts of a file to multiple receivers. User programs may be either event-based or thread-based.

For reference purposes, this implementation was developed for Linux using Gnu C++, the standard template library, and the sockets interface. Portability to other Unix-like systems should be trivial, as should integration with the XIO framework via some simple wrappers. Code will be made available via the OptI-puter CVS archive [24] when it is released.

### 4.2. Scalability

To make our arguments concerning scalability concrete we provide data from the actual CEP implementation. We have designed our system to scale to composite endpoints systems with up to  $10^4 - 10^5$  nodes

in the future. Current systems generally provide less than  $10^3$  nodes in a cluster.

The current prototype uses a single master node to handle the composite transfer, which will obviously be the limiting factor. It maintains global state comprised of the mapping database and a communication channel and status information for each node in the system. This state averages about 200 bytes per host so is not an issue for systems of the scale above. Communicating the required information to all nodes will take four messages totalling about 1KB per node; even on a relatively slow 100Mbps link, we can send this to twelve thousand nodes in one second. Lastly, while determining the optimal resource schedule is complicated (as described above). Our current proportional greedy heuristic performs well, can be calculated in time  $O(n)$ , and does not induce significant CPU load.

## 5. Evaluation

We present a series of experiments which explore the performance, scalability, and capabilities of the CEP approach and our prototype. First, we describe the hardware and software testbeds, then the experiments and results.

### 5.1. Test beds

Tests were performed on three pools of x86-based nodes in a local-area network. The first is comprised of older nodes with 100Mb Ethernet, dual 450MHz Pentium-II CPUs, and 1GB of memory. The second has nodes with 1Gb Ethernet, dual 2.4GHz Xeon CPUs, and 2GB of memory. The third and largest pool (94 nodes) has dual 1.6Ghz AMD Opterons, dual 1Gb Ethernet, an 2GB of memory. All machines run the ROCKS [16] clustering software and RedHat Linux.

We have also performed tests between clusters of nodes across the wide-area network, but we do not present these results. The patterns seen in the data are effectively the same.

### 5.2. Experiments and Results

We present four experiments which explore the CEP prototype’s scalability and performance, efficiency, tolerance of heterogeneity, and the ability to effectively handle data with constrained access.

#### 5.2.1. Composition Efficiency

To evaluate whether we have achieved our composability and scalability goals we test CEP transfers on

varied numbers of homogeneous nodes. Figure 4 graphs the aggregate bandwidth CEP achieves using 1 to 45 nodes (the maximum available) in the third pool, as compared to the theoretical maximum. Each node transfers 2GB and we measure the overall completion time to get the bandwidth measure.

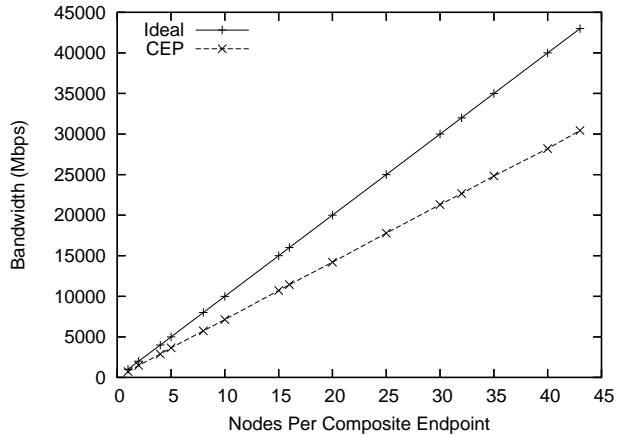


Figure 4. Composition Efficiency

CEP performance increases linearly with the addition of nodes up to the maximum number of nodes available. At the right side of the graph, CEP achieves a peak transfer rate of over 32 Gbps user-level memory to user-level memory— moving 0.8TBytes in just 22 seconds. These results demonstrate the scalability results we expected based on the high-level analysis in Section 4.2.

#### 5.2.2. Efficiency

To determine how efficient a CEP transfer utilizes hardware network resources we use 45 nodes composing a single stream, but consider the network utilization. We also examine host CPU and memory demands. Figure 5 shows network utilization as a percentage of ideal. As before, efficiency is linear, but essentially independent of the number of nodes. CEP achieves approximately 75% resource efficiency in this experiment.

A 75% efficiency leaves room for improvement in future implementations. In particular, we have made certain trade-offs between bandwidth and latency that are obviously not optimal. There are also a few well-understood problems such as excess copies and a lack of critical path optimizations in the current implementation. We are currently addressing these and expect our performance will soon be comparable with the hardware bandwidth. Moving to CPU and memory utilization, during these tests CPU load was similar to that in other TCP-based transfers; so CPU use it not

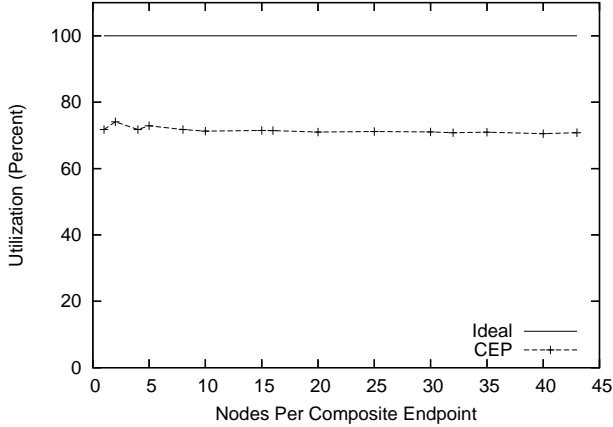


Figure 5. Transfer Efficiency

a bottleneck at these speeds. Memory is proportional to number of flows, and a user-specified amount of pre-cached data; for these experiments it is about 4MB per node.

### 5.2.3. Exploiting Node Heterogeneity

To show the advantages of fully utilizing heterogeneous nodes, we test CEP versus a uniform striping mechanism using nodes taken from the first and second pools. Figure 6 shows the bandwidth achieved for a 1GB transfer. In this experiment we have 16 nodes total, 10 of which are in the slower cluster (the first pool) and 6 of which are in the faster cluster (the second pool). Half of the nodes from each pool are senders, and the other half receivers. We allocate nodes to the transfer from the slower pool until it is exhausted at point “5+0”. Then we allocate nodes from the faster pool.

The CEP and uniform striping schemes give identical performance when the nodes are homogeneous, when faster nodes are added, CEP achieves far better performance, effectively shift to performance growth based on the faster nodes. A simple, uniform striping scheme is limited by the slowest node used. This makes CEP much easier to use. With CEP a user can try a transfer, and if performance is inadequate, simply add resources until performance is acceptable. This addition can be done without concern for details of homogeneity, e.g. whether the additional resources are fast new nodes or a much slower cluster from several years ago.

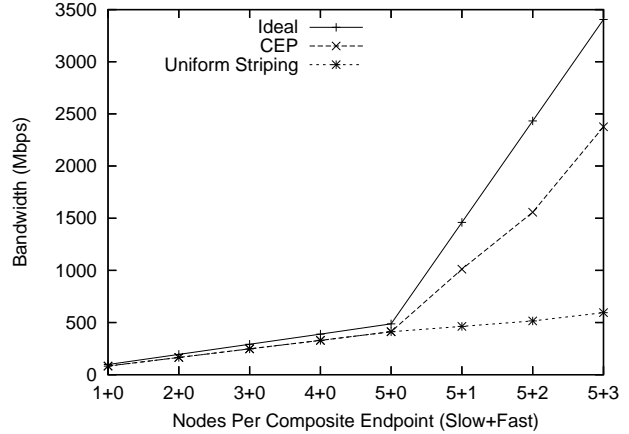


Figure 6. Exploiting Node Heterogeneity

### 5.2.4. Exploiting Data Access Flexibility

Our last experiment explores the performance benefit possible by exploiting CEP’s transfer scheduler or equivalently a CEP transfer’s flexibility in data access. The opportunity is hard to characterize generally, as depends intimately on the data accessibility for a transfer. We explore one example—a fixed set of 8 heterogeneous nodes, four nodes each from the first and second pool, and vary the amount of data each node may access.

Figure 7 shows the results from this test, where the proportion of data each node may access is between  $\frac{1}{n}$ , disjoint data segments, and 1, total overlap. Such variations may be the case when a distributed filesystem is in use or an application’s generation of output data is not homogeneous across all input

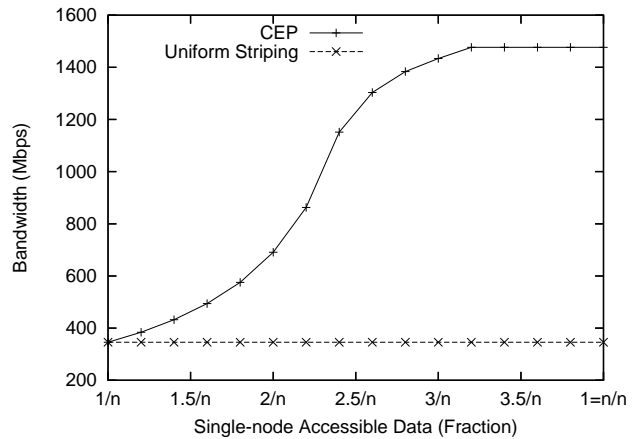


Figure 7. Exploiting Data Access Flexibility

With uniform striping, no matter the data access

flexibility (overlap), performance is the same. With CEP, as the overlap increases, performance does as well, flattening out as the hardware is saturated. This performance improvement is produced by CEP’s scheduling mechanism which uses the data access flexibility to shift work to more powerful nodes until their capacity is saturated. Beyond that point, additional flexibility cannot give additional performance.

Overall, our experiments evaluating the CEP approach and prototype implementation have shown composite performance that scales linearly scalability in the number of nodes (proportional to their speed). In addition, CEP can efficiently exploit heterogeneous nodes, deliver good performance and resource utilization overall. Finally, our experiments show that data access flexibility can be efficiently exploited by the CEP scheduler to move data much faster than single nodes or multiple nodes with regular striping mechanisms.

## 6. Discussion and Related Work

Because the rates of change of technology have inverted the traditional performance relationships of computing and network transmission, CEP addresses a new problem, in which large network flows are faster than the compute nodes used to construct them. Further, CEP supports the efficient composition of heterogeneous nodes to support a single large flow. As a result, there are only a few closely related efforts.

Cluster communication interfaces such as the Message Passing Interface (MPI) [15] support the notion of a communication target, a communicator, which can be used by a collection of nodes as the target for a sequence of messages – an aggregate logical flow. However, MPI, one of the richest of these messaging interfaces, has some limitations. It generally requires uniform or homogeneous hosts, does not provide adaptive scheduling and transport over the wide area, (instead focusing on local parallel machines/a single cluster), is very low-level, among other things. These challenges and others are addressed in CEP by providing global name spaces, adaptive transfer scheduling, and other features as described above. CEP and MPI address different environments with different problems, and their solutions are mutually complementary; each provides a level of functionality that the other can use when appropriate.

High performance file transfer protocols such as GridFTP [1] have proposed (and we have word of working implementations, but know of no published results at this time) of striped file transfers. That is, transfers from clustered nodes with a shared filesystem to other clustered nodes with a shared filesystem. Thus, GridFTP assumes both node homogeneity

and global accessibility of data. To the best of our knowledge, GridFTP implementations only incorporate static striping and multistreaming, roughly equivalent to the “uniform striping” mechanism in Section 5. In contrast, CEP supports node and data accessibility heterogeneity. This means CEP can be used to complement GridFTP and provide additional functionality to applications.

An intriguing body of related work is based on erasure codes and digital fountain techniques [6, 7, 11, 14]. These systems address N-to-1 transfers, typically using many peer-to-peer servers which cooperate in sending blocks of a large video file (typically multiple gigabytes). Because the data being transmitted is static and read-only, it is encoded in advance using erasure codes, and because the transfers are large, the receiver can efficiently choose servers for blocks. In contrast, CEP deals with communication flows that are fully general, does not require any special encoding or structure in the data, and supports short and long transfers, and the general N-to-M scenario.

Finally, distributed systems research has classically focused on problems of coordination, agreement, and fault tolerance as fundamental in any activity involving multiple hosts [4, 8, 12, 13, 22]. The requirements for CEP’s coordination are far less stringent than many of these models, and are achieved through the identification of data bound to a segment of the sequence space in the logical flow. While our approach to fault tolerance is not yet fully developed, it appears that simple solutions are feasible, as CEP transfers will often have redundant sources and receiver targets for data.

## 7. Summary and Future Work

We have introduced CEP, the Composite Endpoint Protocol. It provides a mechanism for allowing multiple processes to join a single logical connection, to be dealt with via a simple and flexible interface, and for scheduling globally optimized network transfers. We have discussed both an actual implementation of this protocol and the theoretical background behind it.

CEP allows one to terminate disproportionately large data transfers, efficiently utilize heterogeneous hardware in an arbitrary  $N$  to  $M$  configuration, and gracefully tolerate a certain class of failures.

In general the performance and scalability has been good. We are currently testing on larger numbers of nodes (above 96) to verify that these trends continue as expected. Also in progress are changes and optimizations that will improve the performance per node, to improve the efficiency numbers given above. Other plans include some engineering and integration work which will make the system even more useful.

## References

- [1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high performance computational grid environments. In *Parallel Computing: Advances and Current Issues*, 2001.
- [2] R. Alpert, C. Dubnicki, E. W. Felten, and K. Li. Design and implementation of NX message passing using shrimp virtual memory mapped communication. In *Proceedings of the International Parallel Processing Symposium*, volume 1, pages 111–119, April 1996.
- [3] A. Bassi, M. Beck, E. Fuentes, T. Moore, and J. S. Plank. The logistical file system: A network file system designed for scalable resource sharing. Technical Report UT-CS-00-469, University of Tennessee, August 2001.
- [4] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.
- [5] R. G. Brown. *Engineering a Beowulf-style Compute Cluster*. Duke University Physics Department, May 2004.
- [6] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.
- [7] B. Cohen. The bittorrent file sharing protocol. <http://bittorrent.com/>.
- [8] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, August 2000.
- [9] E. DeBenedictis and J. M. del Rosario. nCUBE parallel I/O software. In *11th International Phoenix Conference on Computers and Communications*, pages 117–124, April 1992.
- [10] J. E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers in a distributed network. In *Second Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 254–266, 1983.
- [11] KaZaA. KaZaA file sharing network, 2002.
- [12] P. Keleher, A. L. Cox, S. Dwarkadas, , and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proceedings of the USENIX Winter Technical Conference*, January 1994.
- [13] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.
- [14] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of Computing*, pages 150–159, 1997.
- [15] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical report, MPI Forum, 1994.
- [16] NPACI. Rocks cluster distribution. <http://www.rocksclusters.org/>.
- [17] H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing*, 2000.
- [18] L. L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. M. Papadopoulos. The optiputer. *ACM Blueprint for the future of high-performance networking*, 46(11):58–67, November 2003.
- [19] P. State. The lionshare p2p project, 2004.
- [20] V. S. Sunderam. Pvm: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.
- [21] N. Taesombut and A. Chien. Distributed virtual computer (dvc): Simplifying the development of high performance grid applications. In *Proceedings of the Workshop on Grids and Advanced Networks (GAN 04)*, April 2004. Held in conjunction with the IEEE Cluster Computing and the Grid (CCGrid2004) Conference.
- [22] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, January 2002.
- [23] A. J. van der Steen and J. J. Dongarra. Overview of recent supercomputers, 2004.
- [24] Various authors. The OptIPuter CVS archive. [cvs.optiputer.net](http://cvs.optiputer.net).
- [25] Various authors. The OptIPuter project. <http://www.optiputer.net>.
- [26] O. T. Yu and T. A. DeFanti. Collaborative user-centric lambda-grid over wavelength-routed network. In *Proceedings of SC2004*, November 2004.