OptIPuter System Software Framework

The CSAG OptIPuter Team
Andrew A. Chien, Xinran (Ryan) Wu, Nut Taesombut, Eric Weigle, Huaxia Xia, and Justin Burke
optiputer@csag.ucsd.edu
April 12, 2004

Abstract We describe a shared perspective and assumptions looking up and down the OptIPuter system software stack. The discussion clearly defines and justifies the assumptions and model for each perspective, starting with the network hardware architecture, the integrating model of distributed virtual computers and the supporting technologies of group transport protocol, multi-endpoint communication, and shared lambda-grid storage.

1. Network Architecture/Hardware Assumptions

Dense Wavelength Division Multiplexing (DWDM) is increasingly available as an efficient technique to exploit terabit fiber bandwidths, multiplexing large numbers of wavelengths (lambdas) onto a single fiber. The OptIPuter project exploits the ability to dynamically configure these dedicated network paths or lambdas "on-demand". The resulting Lambda-Grids are collections of plentiful, geographically-distributed computing and storage resources, richly-interconnected (10's of gigabits/second) by dedicated lambdas. Lambda-grids are distinguished from traditional shared internet protocol networks by their dynamic configuration and dramatically higher performance and quality of service. Lambda grids can be thought of as real "private networks".

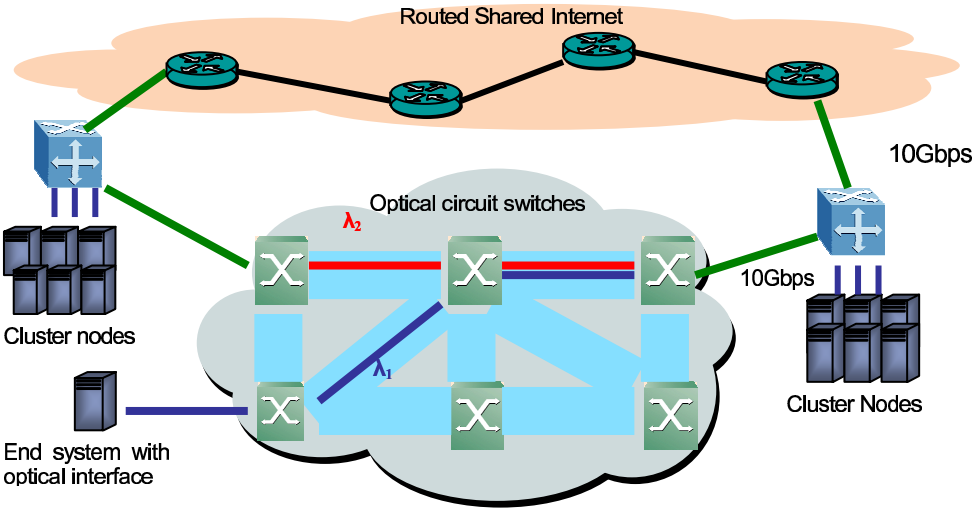

Figure 1. Physical Architecture of the OptIPuter Network

The OptIPuter physical network architecture contemplates two types of lambdas (see Figure 1):

(a) End-to-end dedicated lambdas. These directly connect end computing or storage devices with a private connection running 1-10Gb/s or even more. Such an approach cannot share the connection, and requires each end-node to have an optical interface (which is costly and has low utilization).

(b) Switch to Switch lambdas, shared through fast border packet switch. One or several circuit-switched lambdas may terminate at a shared switch, allowing the connections to be efficiently shared by a set of endpoints. Further, the endpoints need no additional interfaces (optical or otherwise). The advent of cheap 10Gig packet switches and impending cheap 10Gig copper NIC interfaces make this approach attractive. Quality of service can be achieved thru VLAN's or simple over-provisioning (the switches are line speed and non-blocking).

(c) Switch to Switch lambdas, shared through fast border routers. Similar to (b), only a router is used and can be controlled via GMPLS. The challenge here is that router ports running at 10Gbps are prohibitively expensive.

We believe (b)+(c) are the typical cases that will be widely deployed, and reflect the likely integration of optical circuit-switched and packet-switched networks. Optical circuit switching in the OptIPuter network core creates application paths over which high speed packet switches provide the communication "network" flexibility. At the edge, copper ports with cheap, high speed packet switches with some routing services provide non-blocking switching – the equivalent of a direct optical path. The high speed packet switches integrate the OptIPuter and shared IP networks seamlessly, providing easy access to end systems.

Current OptIPuter research networks are a mix of (b) and (c). In Chicago, dedicated lightpaths are dynamically provisioned on demand using routers and VLANs in packet switches. On the UCSD campus network, there are sets of four 1 or 10 GigE dedicated channels over fiber pairs connecting OptIPuter nodes to a Chiaro router (or switching infrastructure). In the CSE annex of the OptIPuter, scenario (b) is deployed across a number of high speed clusters.

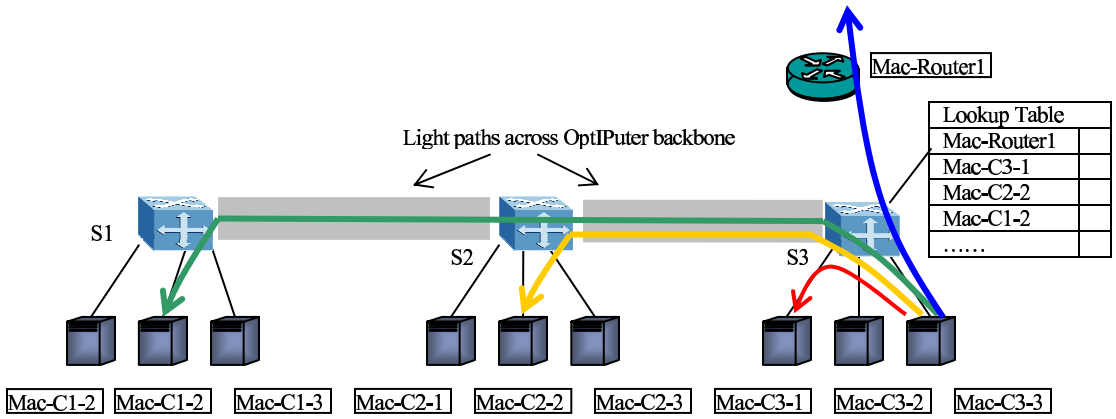

Figure 2. Layer2 connections across multiple edge switches. Four types of connections from cluster node c3-3 are shown with different colors.

The OptIPuter network is operated as a private network in which each cluster is assigned different subnet IP addresses, each set chosen to be disjoining from the "private address" segments of the IP address space. The dynamically configured light path (lambda) between two edge switches acts as a dynamically constructed bridge. Each cluster node may have two network interfaces, typically configured one for each of the routed shared internet and private optical circuit-switched networks. If some interference can be tolerated, a single interface can be configured with two IP addresses, or two interfaces can be used in both networks. The edge packet switches use MAC addresses (e.g. A Summit 400 Gigabit switch supports up to 16,000 MAC addresses) to perform number of key functions:

(a) Switch local cluster traffic among cluster nodes,
(b) Switch shared internet traffic going up into the network and coming down to the cluster nodes,
(c) Deliver packets from local OptIPuter addresses with non-local OptIPuter destination addresses to the appropriate optical 10Gbps connections,
(d) Deliver packets coming from OptIPuter links with local OptIPuter destination addresses to the appropriate interface, and
(e) Deliver packets coming from OptIPuter links to appropriate outgoing OptIPuter links (transit packets).

Lambdas are connected on-demand, which allows applications to manage and control the network resources, e.g., to set-up distributed computing resources and facilitate bulk data transfers. Light path configuration can be static or dynamic. We anticipate lambda setup times of milliseconds, though current systems require much longer latencies. Interfaces to control lambdas are presented as web services which

accept as parameters the desired characteristics and capabilities of a lightpath for provisioning and monitoring.

General assumptions on network and end system hardware for the OptIPuter or Lambda-Grids:

(a) High speed (1Gig, 10Gig, etc.) dedicated links using one or multiple lambdas connecting a small numbers of endpoints (e.g. 10^3, not 10^8), and possibly with long delays (e.g. 60ms RTT from SDSC to NCSA) between sites. Switching and edge structure as described above. The dedicated lambdas have better QoS than shared internet (no jitter, low loss on fiber).

(b) End-to-end network bandwidth matches or exceeds the capabilities of the data processing (computing/I/O processing) speeds of attached systems. The abundant network resources create a relative scarcity of end-system resources, pushing the congestion from internal network links to the endpoints.

(c) Network congestion occurs primarily at the end systems (data buffer overflow) or at the "last switch" where one or multiple high speed streams converge and terminate (e.g. two high speed flows merge toward a single receiver at the switch).

2. Distributed Virtual Computer

Distributed Virtual Computers (DVC's) provide a simple way for applications to exploit Lambda-Grids. DVC's provide natural access to custom communication capabilities by expressing it as connectivity amongst end resources, and naturally bundle it with grid resource specification. In addition, the shared namespace across the resources that DVC's support provides a natural way to expose the unique communication capabilities provided by optical networks (such as optical multicast).

More precisely, a DVC is a collection of Lambda-Grid resources, aggregated from multiple administrative domains and even spanning wide-area networks. These Lambda-Grid environments consist of dynamic, heterogeneous, restrictively shared, and large numbers of resources connected by dedicated lambda connections. DVC's provide users and applications with a powerful set of abstractions that hide the complexity and management of underlying software and hardware infrastructures. An application need only consider a tightly-coupled resource pool managed under a single security and control domain. Key DVC services for users and applications include:

1. Naming and namespace management: Resources are assigned unique site-independent names. Set operations on these names enable convenient expression of relationships amongst them, including level of trust, communication schemes, etc.
2. Security Management: Security domains can be formed across sets of resources. Finer-grained trust relationships and access control policies can be defined.
3. Resource Management: Resources can be dynamically added and released from a DVC. Resource scheduling and monitoring are provided to support fault-tolerance and resilience of the DVC.
4. Communication Management: Varied communication paradigms, mechanisms, and protocols can be incorporated into a DVC (e.g. multicast, distributed shared memory, etc.).
5. Event Monitoring: To support robust applications, event subscription/notification and asynchronous message services provide notification of changes in network and resource status.
6. Job Execution Management (optional): Batch, interactive and large-scale distributed jobs can be submitted to execute on the DVC. These jobs are scheduled, mapped to run on appropriate resources, and monitored to ensure efficient and reliable execution.

To implement these services, we will rely on underlying Grid technologies, leveraging the Globus Toolkit, the Network Weather Service, and emerging high-performance transport protocols (e.g. GTP, RBUDP, UDT, etc.) for respectively resource discovery and allocation, resource monitoring, base security infrastructure, and data movement, exploiting them to provide more usable and application-oriented services. DVC abstractions will be implemented and presented as web services using (WS-RF) specifications. This implementation approach enables easy use and fast deployment of the OptIPuter software with existing web tooling that can be run on many platforms.

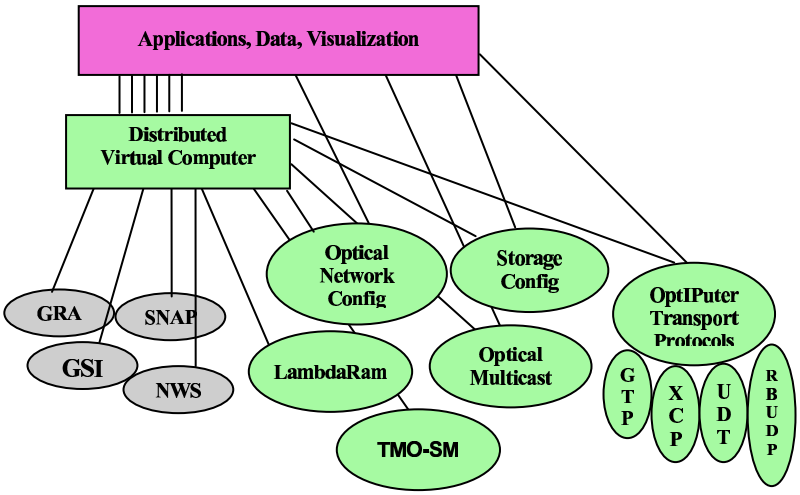

Figure 3. Distributed Virtual Computers build simple application execution environments, but present a web services interface which allows direct access to lower level primitives.

Example: Using a DVC

To create a DVC (or to add resources), a user creates a specification of the desired resource configuration and passes it to the DVC resource discovery and selection service. In response, he receives a physical resource configuration that matches all the requirements in the specification. If not satisfied, he can modify his specification and try again. Once satisfied with the returned configuration, the configuration is passed to the DVC resource binding service which binds the selected resources into the DVC. The user can further organize the allocated resources into sub-domains, specifying their trust relationships, and creating complex communication structures amongst them. When done with this process, the user is given a DVC descriptor which can then be used to for direct access to the DVC resources, and used to submit applications, computations, etc.

Enhanced DVC: Real-Time Distributed Virtual Computer

Basic DVC's have controllable communication properties and thus when combined with real-time schedulers, can provide tight performance guarantees. A Real-time DVC is a specialized DVC which combines predictable communication and resources with distributed real-time scheduling of communication and end resources. An example of this would be to combine the UCI Time-Triggered Messages and Objects (TMO) system with the base DVC technology, producing a system with both a high level programming interface and real-time properties.

2.1. Integrating Group and Multi-Endpoint Communication

Lambda-Grids inspire new communication patterns, including multipoint to point, multipoint to multipoint, and even multicast structures. With a shared namespace, DVC's enable expression of complex communication structures as well as requirements, including performance, security, etc. Novel protocols involving more than two endpoints such as GTP, Multi-endpoint, and Multicast can all be expressed. Even distributed state models such as LambdaRAM can be expressed.

2.2. Integrating Shared Grid Storage

While many data Grid systems are formulated on sharing through web services or other forms of high level mediation, low-level integration can yield much higher performance. DVC's allow storage resources to be securely into a DVC and directly accessed with high performance. Thus, a wide range of wide-area storage aggregation and sharing techniques are possible – analogous to those feasible on a SAN with very long links. For example, in a DVC, a remote disk could be accessed as if it were local. A set of 28 remote disks could be bundled together into a stripe group for a software RAID.

3. Dealing with Networks Faster than Endpoints

3.1. Group Transport Protocol

Large-scale computation and data sharing in Grids and peer-to-peer applications is driving an evolution in communication patterns from point-to-point connections to multipoint-to-point and multipoint-to-multipoint structures. Within a DVC which defines a set of endpoints and sets up optical resources, the goals of Group Transport Protocol (GTP) is to efficiently manage endpoint contention. Ideally, GTP would match the performance of centralized omniscient scheduling of distributed group (multipoint-to-point, and multipoint-to-multipoint) communication. That is, to manage converging flows whose total capacity exceeds that of any single endpoint. More specifically, GTP's goals are to:

(a) provide efficient (high throughput and low-loss) multipoint-to-point and multipoint-to-multipoint data transmission,
(b) guarantee intra and inter protocol fairness amongst flows. This also includes the issue of how high speed flows could co-exist with traditional TCP (and its variants), and
(c) respond quickly to flow dynamics (when new flows join or some flows terminate);

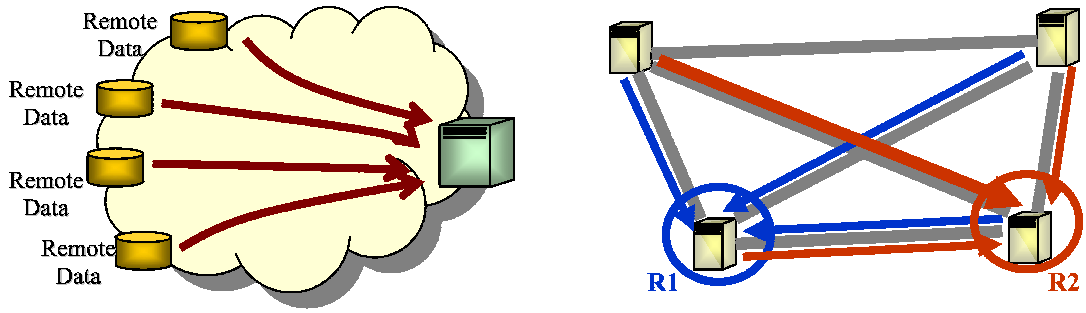

Figure 4. Multipoint-to-point communication

The Group Transport Protocol (GTP) is a receiver–driven transport protocol that exploits information across multiple flows to manage receiver contention and fairness. Key novel features of GTP include

(a) Request-response based reliable data transfer model, flow capacity estimation schemes;
(b) Receiver-oriented flow co-scheduling and max-min fairness rate allocation;
(c) Explicit flow transition management;
(d) Multipoint-to-multipoint, with coordination among multiple endpoints;
(e) Coordination between TCP flows with GTP flows.

Preliminary measurements from an implementation of GTP show that for point-to-point single flow case, GTP performs as well as other UDP-based aggressive transport protocols (e.g. RBUDP, SABUL), achieving dramatically higher performance than TCP, with low loss rates. Our results also show that for multipoint-to-point case, GTP still achieves high throughput with 20 to 100 times lower loss rates than other aggressive rate-based protocols. In addition, simulation results show that unlike TCP, which is unfair to flows with different RTT, GTP responses to flow dynamics and converges to max-min fair rate-allocation quickly.

GTP will be available through the following interfaces:

a) In form of XIO (a unified I/O interface for grid applications) to support grid applications
b) A set of GTP API to support general applications.
c) A service within DVC, to support applications running on top of DVC.

3.2. Multi-Endpoint Communication

While GTP addresses convergent flows whose aggregate speed give rise to endpoint congestion, the speed of dedicated optical connections alone can also induce endpoint congestion. For example, how do you terminate a 100Gbit flow? Or more immediately, how do you terminate a 20Gig flow with a cluster of

1Gbit machines. This is a specific instance of the general issue that in the future network resources will be plentiful and that the bandwidth available will be larger than any "average" host can terminate. By spreading the communication overhead on to multiple hosts, an arbitrarily large flow can be terminated, or even extremely powerful nodes can spend a greater percentage of their time performing more useful computations. This allows us to treat clusters as a single endpoint or "supernode" in the DVC, efficiently terminating a high-speed lambda shared among multiple hosts. Of course, multi-endpoint techniques can also be viewed as increasing the endpoint capacity available, allowing protocols such as GTP to go even faster.

The idea behind multi-endpoint communication is to terminate a single flow split across a set of hosts which may be homogeneous or heterogeneous. One difference is that a multi-endpoint/single flow approach need not guarantee fairness between hosts and can maximize performance globally. Consider the case when several heterogeneous hosts have access to a common data source and wish to move a large amount of data from this source over the network. In a multi-endpoint connection we can allocate the resources proportionally, even if this means some nodes transfer more data than others. This reallocation of load will increase global efficiency. Note that fairness between a multi-endpoint flow and other flows can still be guaranteed.

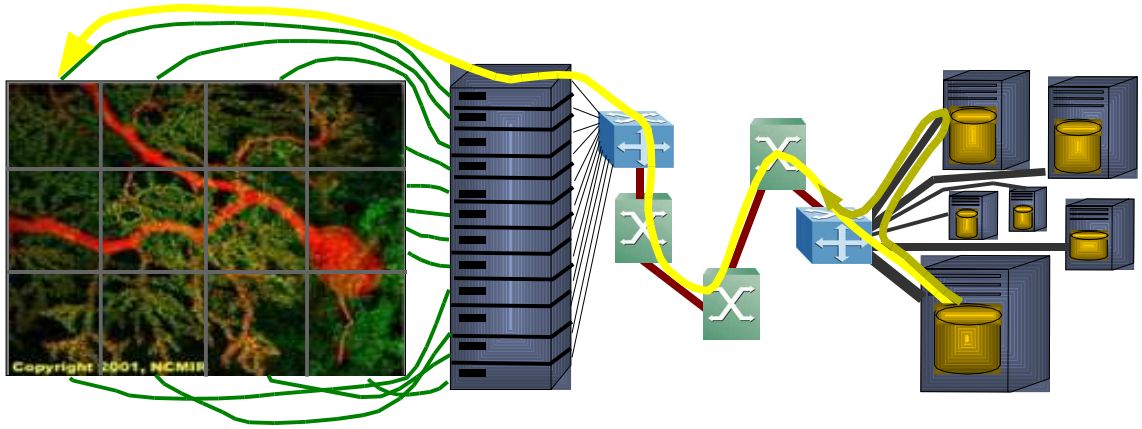

Figure 5. Flexible data traffic in a many-to-many connection. Heterogeneous hosts may take part unequally in a logical connection and reroute traffic amongst themselves.

The multi-endpoint concept also provides several features that cannot efficiently be implemented within a point-to-point model. By allowing trusted hosts to explicitly cooperate as one end of a connection, we can improve performance under a number of measures (bandwidth, efficiency, fairness, etc) by exploiting:

- Flexible mappings between senders and receivers, such as any-to-any communication
  - With point-to-point links the number of connections required is $O(n^2)$.
- Adaptive load balancing on the basis of flexible mappings
- Multipath routing within a single logical connection to exploit multiple routes between hosts.
- Quick rerouting to compensate for node failures/recoveries, congestion events, etc.
- Simple abstraction and API.

The simple abstraction consists of an API in two parts: first a specification of the desired connection structure, and second a set of socket-like commands to act on the connection (read/write, open/close, add/remove host, etc). The structure definition will indicate which sets of hosts will communicate (either a static mapping or allowing dynamic configuration) and other metadata for the connection. The commands will take this structure, analogous to a socket identifier, and perform the appropriate operations requested. This API will expose the desired structure for the communication at exactly the right level: it will be visible when mapping communication onto actual (or virtual DVC) network resources, but hidden so that application designers need not deal with the "nuts and bolts" of the implementation. Note that the problem

of resource naming and location is left to another layer (such as the DVC, DNS, etc) while resource allocation is partially handled here via automatic mechanisms in the protocol.

The multi-endpoint model supports message-based communication. The parts of these messages may traverse the network using a variety of transports, including UDP, TCP, FAST, or others. Choice of protocol depends on their availability and the desired connection properties. For example, if we have a simple two-to-one sender-receiver mapping we may use TCP for these static connections, and a simple reliable UDP to shuffle data between hosts at one endpoint to reroute packets after a congestion event affects one of the TCP connections. In this way we may leverage the best aspects of each protocol and provide any desired fairness/congestion/latency/etc model desired in a simple manner.

4. OptIPuter Shared Grid Storage

Data-intensive, distributed applications require the development of new high performance storage systems. These applications include large-scale scientific computing, data mining, and interactive visualization, and have novel requirements: 1) large datasets, often in terabytes to petabytes; 2) due to collaboration, distributed data generation, large dataset size, or reliability or performance purposes, the datasets are typically distributed across multiple storage sites; 3) low latency requirements for datasets, especially for high-performance computing or interactive applications. Lambda-Grids provide an opportunity to explore new distributed storage architectures. Most data grid systems are formulated on data sharing through web services or other forms of high level mediation (database federation or GridFTP). DVC's allow storage resources to be accessed directly and securely, enabling the use of SAN-like and a wide range of novel new techniques for distributed storage access. For example, in a DVC, a remote disk could be accessed as if it was local or a set of remote disks could be bundled together into a stripe group for a software RAID. The RobuSTore Storage System being developed at UCSD is a grid storage system exploiting the capability of multiple shared storage devices connected by the OptIPuter high-performance optical networks.

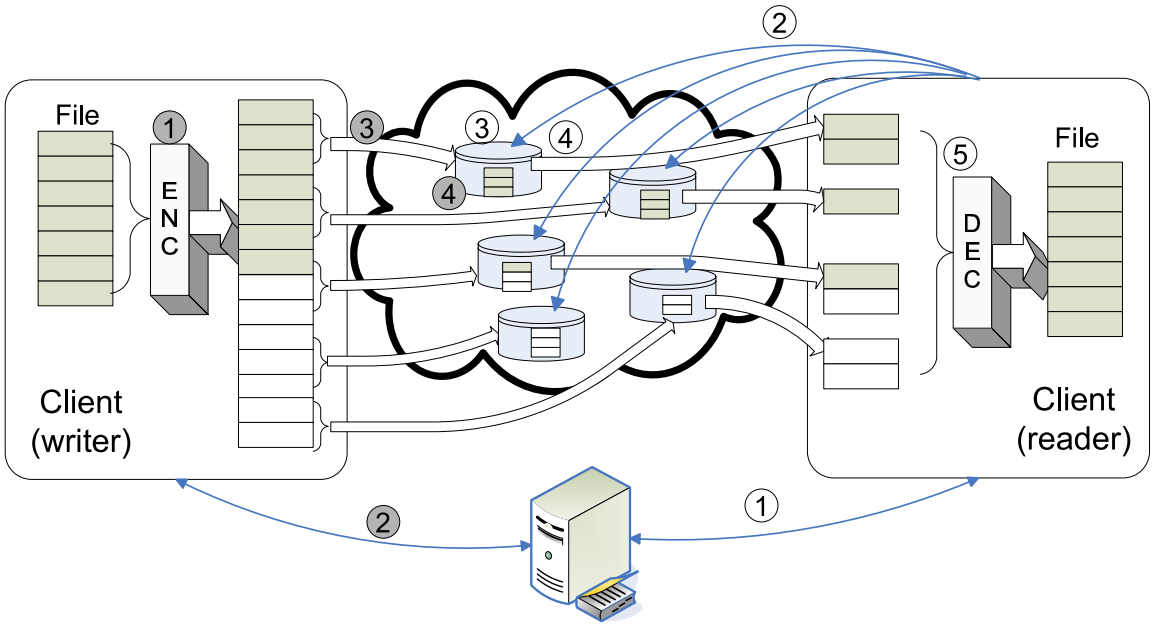

Figure 6. Data access in the RobuSTore System

4.1. Introducing Redundancy for Performance

The RobuSTore system uses concurrent access to multiple disks to increase both absolute performance and performance robustness. Traditional distributed file systems are optimized for low-bandwidth networks, but OptIPuter environments have abundant bandwidth. This pushes access bottlenecks to the storage devices where parallelism can be combined with redundant encodings to improve performance

characteristics. Exploiting redundancy, a client can reconstruct the desired data once it has received some minimal number of data blocks (as specified by the encoding scheme).

When data is stored in RobuSTore, each large block is encoded using an erasure code scheme; each original K data blocks becomes N coded data blocks (where N>K). These coded data blocks have the property that any K(1+ε) blocks are sufficient to reconstruct the original data, where ε is either zero or a small fraction, depending on the coding algorithm. The coded data blocks are distributed to multiple storage servers in concert with a metadata server. When a client reads, it first obtains the locations of the coded data blocks from the metadata server; the client then requests the coded data blocks in parallel from multiple storage servers. Once the client gets K(1+ε) coded blocks (of the possibly N requested coded data blocks), it can reconstruct the original data. After requesting more than K(1+ε) blocks, the RobuSTore system will reconstruct the data from the first K(1+ε) data blocks returned, exploiting the variability in response time to avoid waiting for the slowest responder.

RobuSTore can also use redundancy to improve performance robustness. In a shared environment, each storage device may have multiple request streams (from different clients) and hence have dynamic response time and throughput. The above scheme can reduce the impact of dynamic loads by isolating the access performance from the slowest (more loaded or slower device) responders. Placement and fetching algorithms can also be tuned based on a wide range of other dynamic or static storage device characteristics such as capacity, throughput, workload, availability, allocation policy, network capacity, etc.

4.2. Security Requirements

Clients in the RobuSTore system need to access the shared storage devices, so the client has to be trusted by the remote storage devices, or the set of operations enabled must be safe. We will use interfaces provided by DVC to access remote storage devices. In this case, the remote storage devices appear as SAN nodes to the clients. A wide range of schemes are being considered.

4.3. Network Requirements

The RobuSTore system depends on high-performance network communication since it speculatively reads data from multiple devices in parallel. The following network properties are critical:

(a) Abundant bandwidth, enabling aggressive prefetching.
(b) Efficient multipoint-to-point and multipoint-to-multipoint communication, supporting efficient data reads across striped or redundantly encoded data (ensured by GTP).

Acknowledgements

Supported in part by the National Science Foundation under awards NSF EIA-99-75020 Grads, NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from Hewlett-Packard, BigBangwidth, Microsoft, and Intel is also gratefully acknowledged.