Mediaflux® Livewire

Transfer at the speed of light

Abstract

Over the past thirty years the amount of data that is transmitted around the globe has grown enormously, commensurate with the amount of data we produce – now collectively measured in 10's and 100's of zettabytes. The fastest way to move very large amounts of data was, and often still is, to physically move storage from one place to another. That requires a lot of coordination and is ultimately not scalable.

Our ability to digitally transmit data over vast distances is limited by the speed of light (2.998×10^8 meters per second in a vacuum). If we had perfect photo-couplers and electronics that had zero latency, then transmitting a single datum between Melbourne in Australia and New York in the United States – approximately 16,662 kilometers – would take around 56 milliseconds one way, or double that for the return journey – a theoretical asymptote of 112 milliseconds. Of course, the electronics and photo-couplers are not perfect so a return trip between those two cities currently takes around 232 milliseconds. To illustrate the effect of that latency: at that rate if you were to wait for acknowledgment, you could type fewer than 4 characters per second.

The throughput of the network pipes is increasing: from 110 baud modems in the 1950's to networks capable of transmitting 100 gigabits per second between continents – up to 11 gigabytes per second, or terabytes in a few minutes. One terabit per second networks will follow and at that rate, it should be theoretically possible to transmit 110 gigabytes per second from point A to point B – which equates to around ten terabytes in a few minutes.

The challenge – and the objective of the Data Mover Challenge 2021 – move as much data from A to B as efficiently as possible, regardless of the distance and the constraint imposed by the speed of light. Of course "efficiency" is not just a function of the network speed - it's a function of the end-to-end process.

Concepts

The problem of transmitting data from A to B can be reduced to the following sub-systems:

The **Source** subsystem is responsible for the provision of data to be transmitted. This will involve the timely scanning and reading of data from source system(s). This is typically external storage but could also be dynamically generated data provisioned from memory or other systems.

The **Destination** sub-system is responsible for writing data to objects. A Destination is first classified by a protocol (type of) which is one of: a) **file** – a file system that is external to Mediaflux. b) **null** - requests the data read from the network be "dropped on the floor", or c) **asset** – the destination is the asset sub-system within Mediaflux .

The supported systems for source and destinations are: a) memory, b) disk through locally mounted POSIX file-systems, c) storage presented as NFSv3, d) storage presented as SMBv3, e) S3, including Amazon, f) Amazon Glacier, g) Azure, h) Google Cloud Platform or i) tape.

Mediaflux has sub-systems for scanning and processing source data. The scanner is multi-threaded – it will recursively walk down a "directory" tree and will split off with a separate processor if there is still a thread available . The scanner has been used to efficiently process data sources with many billions of files. The multi-threading in the scanner improves performance with regular file systems. It is particularly important when the source is higher latency such as cloud-based storage.

The network transmission component is highly parallelized and able to deal with a single very large file or many very small files. It creates a native protocol tunneled via HTTP/S over TCP/IP. That means the system can use network infrastructure that is commonly available to all people and organizations.

The Problem

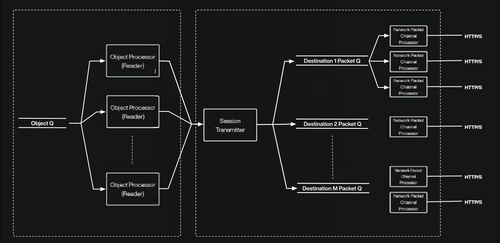
We framed the problem as follows:

> Given an arbitrary network N capable of transmitting data at some rate R between two points A and B with latency L, then transmit at the maximum rate R regardless of the scale of L and regardless of the composition of the data. Any approach must scale to achieve the rate R, regardless of the magnitude of R.

That is, transmit as much data as possible in the shortest possible time – fully utilize the available bandwidth, regardless of the latency and the composition of the storage systems.

The speed of transmission should be unaffected by the number and size of the data to be transmitted - it should work equally well for billions of small objects / files as for a few large objects / files.

Transmission time should include the time it takes to prepare data for transmission for arbitrary data sources.



Those networks can be enhanced independently of, and without impact on, the transmission software - the simplest example of this is changing from IPv4 to IPv6, or transmitting IP over fabrics such as Infiniband - those can be changed / enhanced without affecting Mediaflux Livewire.

The set of objects to be transmitted might be constructed from arbitrary query predicates. For example, "transmit all objects - that fall within some arbitrary geospatial extent - from A to B", or "transmit all microscope images that have been tagged as containing tumors of certain types and convert those images to some other format for interchange before transmitting."

Transmission might be ad-hoc or a part of a workflow, including being triggered when there is a change in the metadata associated with an object; for example, an object may be automatically transferred as soon as it is been marked as PUBLISHED.

The storage sub-systems are managed transparently by Mediaflux. The storage may be high-latency such as tape, or Amazon Deep Archive. Mediaflux will batch pipeline the recall of the data and will transmit as soon as the data is available - in the case tape, it will request large numbers of files to be recalled and then transmit each file as soon they have been individually recalled. One Mediaflux system has been clocked moving data at 65TB per hour off tape. That is, Mediaflux will manage the overall transmission pipeline, including the management of storage at each end.

Mediaflux will combine disparate storage into a single global namespace. This improves aggregate performance and also allows the transmission of data that would exceed the limitations of any single storage system.

The systems can be combined into a cluster of cooperating nodes . Those nodes can scale out to increase the aggregate bandwidth to terabytes or more of data per-second that can be centrally managed.

Design Principles

We set the following design principles:

1. Use TCP/IP, not UDP

UDP is more difficult to operationally manage than TCP/IP, including QoS management. We had previously shown that with sufficient concurrent circuits and a large enough TCP window size, TCP/IP can fully saturate a given network link. TCP/IP can piggy-back on standard network ports (HTTP/S) without requiring "special" ports to be opened-up.

2. Leverage Linux kernel network optimizations

We have previously experimentally determined that when the host operating system is Linux: a) we should not set the buffer size and allow the kernel to auto-tune, or b) we do set the buffer size and ensure the kernel limits allow that buffer size, otherwise we'll end up with a small buffer size and will have disabled the auto-tuning in the kernel – the worst possible outcome.

3. Maximize the overlap and parallelization of I/O

Blocking operations should be executed in parallel rather than sequential. Every aspect of the pipeline should be parallelized from the source storage, network and destination storage. The overall transmission speed will be limited by the slowest component.

4. Do not use zero-copy

We could use zero-copy in the kernel to bypass copying data into user space - which does increase throughput performance, but only in a few specific cases – e.g. the source system is file and the destination is a socket. However, this is too specific as Mediaflux supports arbitrary data sources, many of which are not files.

Platform capabilities

High-Performance Parallel Scanning – high-performance, parallel file scanner – designed to handle billions of files.

Parallel Network I/O – clustered systems scale out to terabytes per second of I/O or more.

High-Performance Database – native database engine has support for trillions of objects.

Authentication, Authorization, Auditing – operating on all networks from unclassified through to top-secret environments. Role-based authorization.

Access Control and Auditing – automatically remove metadata that should not be transferred to another location. Every movement of data is audited.

Support for 1000's of Petabytes of Data – 2^{63} objects per system, 2^{63} bytes per object per system, arbitrary number of systems connected in a distributed federation.

End-to-End Encryption – strong encryption during transmission and storage. Individual encryption key management.

All Data Is Check-summed – all data is check-summed (transmission and storage) for fault detection and recovery.

Global Object Identification – uniquely track every object as it moves around the globe.

Span on Premise and Cloud – distributed architecture for moving data between premises, between cloud, or any combination.

Duplicate File Detection – identify and eliminate duplicate data.

Chained Workflows – automatically initiate workflows on receipt of data. E.g. send data to a compute resource.

For more capabilities, scan the QR code below.

ARCITECTA®
OPERATING SYSTEMS FOR META+DATA

www.arcitecta.com
livewire@arcitecta.com