Managed File Transfer as a Cloud Service

Brandon Ross, Engin Arslan, Bing Zhang, and Tevfik Kosar

Abstract Applications in science and industry have become increasingly complex and more demanding in terms of their computational and data requirements. Sharing and disseminating large datasets has become a big challenge despite the deployment of petascale computing systems and optical networking speeds reaching into the hundreds of gigabits per second. Having high-speed networks in place is necessary but not sufficient for achieving high data transfer rates. Being able to effectively use high-speed networks is becoming increasingly important for cloud computing. Cloud-hosted managed file transfer (MFT) applications simplify high-performance data transfer in the cloud by efficiently utilizing underlying networks and effectively coscheduling concurrent data transfer tasks. This chapter explores the concept of MFT in the cloud and looks at the design and implementation of one such MFT system—StorkCloud—as a case study.

1 Introduction

Data analysis is now more important to industrial and scientific research than ever before. As its importance has grown, so too has the need to share and analyze the very large datasets that are now commonplace in research. Large scientific experiments, including environmental and coastal hazard prediction [15], climate modeling [13], high-energy physics simulations, and genome mapping [7] generate petascale data volumes on a yearly basis [11]. Data collected from remote sensors and satellites, dynamic data-driven applications, and digital libraries and preservations also produce large datasets [9, 20].

Today, a large portion of big-data processing and analysis is performed with the help of cloud services. Economies of scale have made outsourcing the processing of this data to distributed cloud services a popular alternative to deploying and maintaining similar services on-site. However, this change in system architecture from local to distributed has not come without its complications. The importance

B. Ross • E. Arslan • B. Zhang • T. Kosar (✉)
Department of Computer Science & Engineering, University at Buffalo,
The State University of New York, 338J Davis Hall, Buffalo, NY 14260, USA
e-mail: bwross@buffalo.edu; earslan@buffalo.edu; bingzhan@buffalo.edu; tkosar@buffalo.edu

X. Li, J. Qiu (eds.), *Cloud Computing for Data-Intensive Applications*,
DOI 10.1007/978-1-4939-1905-5_16

of data analysis to modern research and industry necessitates global collaboration and sharing among many organizations, which results in frequent large-scale data movements across widely distributed sites.

Several national and regional optical networking initiatives such as Internet2 [4], ESnet [3], XSEDE/TeraGrid [21], and LONI [18] provide high-speed network infrastructure for sharing this data, and recent developments in networking technology make high-speed optical links reaching up to and beyond 100 Gbps in capacity available [1] for members of the scientific community. Yet despite the availability of these high-speed wide-area networks and the use of modern data transfer protocols designed for high performance, data transfers in practice often only attain fractions of their theoretical maximum throughput. Indeed, many organizations even resort to sending their data through shipment services such as UPS or FedEx rather than moving data through the Internet [10]. This inability to fully utilize network infrastructure and easily move data is a contributing factor to the "data deluge" we are now in the midst of.

From this, it is apparent that having high-speed networking infrastructure in place is a necessary but not sufficient condition for performing high-speed data transfers. Being able to effectively use these high-speed networks is increasingly important for wide-area data replication and federated cloud computing in a distributed setting. Doing so requires effectively coscheduling and dynamically optimizing data replication tasks in a way that maximizes transfer efficiency and minimizes resource contention.

A number of cloud services have been established that aim to do just this. These cloud-based **managed file transfer** (MFT) applications offer a solution to the problem of making effective use of networking infrastructure for cloud data transfer by providing a service-based approach to the planning, scheduling, monitoring, and management of data placement tasks. Their position as a centralized and dedicated data transfer coordination system allows these services to offer increased data transfer reliability and security, effectively coschedule concurrent data transfer operations, and deliver performance improvements through capabilities such as connection caching, scheduled storage management, and end-to-end throughput optimization for broad ranges of data-intensive cloud computing applications and storage systems. This chapter will introduce the concept of MFT and describe its role in today's widely distributed cloud computing ecosystem.

StorkCloud, developed by the Data Intensive Distributed Computing Lab at the University at Buffalo, is one example of an MFT system, and will be used as a case study of such systems in this chapter.

This chapter will also discuss various data transfer optimization techniques that can be employed by MFT applications. Techniques such as command pipelining, data channel parallelism, concurrent file transfers, and other techniques which can mitigate the factors which lead to poor network utilization will be discussed, along with algorithms and heuristics which can be used in centralized scheduling to increase long-term transfer throughput.

2 The Problem of Data Insolubility

The complementary roles of processing and storage have a long history in the field of computing. The flow of data from storage medium to locus of processing and back again could be said to be computation's defining feature. Given the importance of computation to all modern technology, it's no surprise that bigger data storage and faster processing systems are always in demand.

With the advent of cloud computing and the benefits it brings, outsourcing data storage and processing to the cloud is a very compelling option for many organizations versus providing such services on-site. However, the transition to the cloud has not been without its difficulties, particularly where data needs to be moved to, from, and especially within the cloud—activities that are becoming more common as both storage media and loci of processing disappear into the cloud.

Most storage services focus on data storage primarily and offer simple—usually proprietary—data transfer schemes solely for the purpose of allowing clients to store and retrieve data directly. As their primary focus is on storage, they typically do not concern themselves heavily with transfer performance, optimization, scheduling, or the details of the underlying transfer protocols. Migrating or copying data between cloud storage services also poses a challenge due to the proprietary data access solutions employed by many cloud storage providers. Indeed, storage services may even have incentive to keep it that way if it makes it more difficult for a client to switch to a competitor.

Staging data into cloud data processing centers from an external storage system poses similar challenges. These processing services often either provide their own proprietary data staging schemes, or otherwise require users to manage data placement themselves. In most cases, unless the cloud storage service and computation service are both managed by the same provider, the exercise of moving data from one site to the other is a detail left to the client. Cloud computation services, like storage services, likewise do not usually concern themselves much with transfer performance. The data staging process is frequently treated like an afterthought, or at least something outside of the provider's domain of concern.

This insularity has unfortunately led to an ecosystem in which users must make sacrifices in order to reap the cloud's benefits. Either they forsake flexibility and constrain their applications to services explicitly designed to exchange data—usually limited to a single provider or a small handful of cooperating ones—or else they take on the burden of custom rigging their own data transfer solution, as well as guaranteeing its reliability, security, compliance to regulation, availability, performance, and future maintainability. Is this choice between inconvenience and inflexibility not antithetical to the nature of the cloud, whose very benediction is supposed to be the flexibility and convenience it offers over traditional solutions?

2.1 Solutions

There are two immediately apparent solutions to the problem of data insolubility in the cloud:

1. widespread agreement on and subsequent deployment of an open standard for data placement, or
2. the institution of services which manage data transfer on clients' behalf.

The first solution has a number of issues. Whose standard do we use? Many competing standards, open and otherwise, already exist and have not seen widespread adoption. History tells us that new "universal" standards intended to replace competing standards often turn out to be just another competing standard, exacerbating the problem. Furthermore, as was mentioned, cloud storage providers have economic incentive to continue using proprietary standards. The inertia required to change the ecosystem at the provider level likely makes this approach infeasible, at least in the short term.

The second solution is perhaps a more feasible approach, and the one that seems to be gaining the most traction. Cloud-hosted MFT applications which aim to provide large-scale data placement between remote sites have begun to appear as recently as 2010. These services provide a "software as a service" (SaaS) approach to data management, and purport to offer a solution to the problems of vendor lock-in and incompatible APIs (application programming interfaces) that are responsible for the data insolubility problem we face today.

At the very least, MFT in the cloud provides a stopgap measure to mitigate these problems. However, these services also confer a wide range of potential benefits and new possibilities, as we shall soon see. Indeed, the future may very well find MFT a valuable and permanent resident in the cloud ecosystem.

3 Managed File Transfer

Before going any further, a clarification must be made regarding the meaning of the "MFT" as used in this chapter. MFT has been used in the past to describe any kind of solution which facilitates large-scale secure data transfer over wide areas and allows organizations to centrally control and monitor data transfers as they take place. This includes hardware and software installed on local sites to manage the movement of data into and out of the physical premises in which the data is stored or processed.

This chapter uses the term MFT to describe such a solution *provided as a cloud service*—that is, entirely off-site and without special provisions at the endpoints. Each instance of "MFT" in this text could very well be replaced with "MFT as a service" or "MFT in the cloud", however this would be considerably more onerous to read, and should be obvious given the subject of this book. It should be assumed that when "MFT" is used in this chapter it is referring to MFT as a cloud service unless otherwise noted.

A distinction must also be made between MFT applications and other types of services which are commonly identified as data or file transfer services. Many so-called data transfer services are temporary data storage systems used as an intermediary for exchanging files between individuals. In these systems, file data is uploaded to the hosting system by the sender and stored there for a limited amount of time or until it has been retrieved by the recipient. Such services are mostly indistinguishable from cloud storage services, with the difference being that files are hosted temporarily with the intention of being downloaded only by a small number of recipients.

An MFT application, in contrast, is used to schedule and coordinate data transfers between distributed endpoints, in a sense acting as "glue" for data storage and processing systems in the cloud. MFT applications do not necessarily act as a physical intermediary for data, though they are of course not excluded from doing so. They may instead communicate with remote storage systems using protocols understood by the end systems, and negotiate direct system-to-system transfers without data flowing through the MFT system. Such transfers are called **third-party transfers**. An MFT system might offer temporary data hosting (sometimes called "data parking" in this context) in order to offer improved transfer reliability, though this is not necessarily the case.

MFT introduces a number of benefits over *ad hoc* data transfer implementations. For one, MFT services can offer asynchronous "fire-and-forget" functionality, where a client can specify an immediate or future data transfer and delegate reliability and performance concerns to the MFT system. The system will monitor transfer progress and deal with issues as they arise. Clients can check on transfer progress through the system, cancel or reschedule transfers if they so desire, and be notified by the system when the transfer completes or if it cannot be completed.

MFT applications can also provide support for numerous transfer protocols, and even perform translations between otherwise incompatible protocols by acting as an intermediary. This allows existing storage infrastructure to be used without needing to reconfigure end systems to "speak" the same protocols.

MFT applications may also offer suites of transfer performance optimizers to algorithmically tweak transfer settings and schedule transfers in order to minimize conflicts and avoid network congestion. Such optimizers can take into account transfer priority and user-specified deadlines. An MFT system can also maintain a historical transfer performance database for different systems to better estimate transfer completion time and schedule transfers to meet deadlines.

An MFT system should also be able to access remote system directory listings and file metadata for the systems and protocols it supports and present it in a unified format. In most cases, the capability of accessing remote metadata is a prerequisite for performing remote data transfers in first place, making the provision of directory listing information by an MFT system straightforward. In this chapter, a service offering such functionality is referred to as a **directory listing service** (DLS). Such services can be used for the development of interactive interfaces for browsing file system hierarchies on remote endpoints. With user interactivity in mind, a DLS may also take measures to improve the responsiveness of metadata and listing access operations by, for example, caching and prefetching directory listings.

As is typical of cloud services, an MFT system should offer both a graphical (typically web-based) front end for users to interact with the system, as well as an API for allowing programmatic access to the system's services and the development of third-party client applications. The availability of a machine-accessible API is important for an MFT system to serve its role as an in-cloud connectivity layer between distributed cloud-based data storage and processing services.

Lastly, MFT applications can relieve users of the burden of having to manage transfer security themselves. Such applications can securely manage remote system credentials on a per-user or per-organization basis, allowing clients to schedule regular secure transfers and reduce the frequency of credential exchanges with the system. MFT applications may also take the steps necessary to comply with regulations regarding secure and private transfer of sensitive data, such as those put forth in HIPAA.[1]

An MFT application has the advantage of not needing to invest heavily in either storage or computation resources. Instead, the only critical resource in an MFT system is network connectivity, allowing for inexpensive, geographically distributed deployment of the system.

3.1 Examples of MFT

A number of MFT services exhibiting some or all of these features already exist and are well-established in the cloud ecosystem. Even at the time of writing this list is not complete; these are only a few examples.

Globus[2] is a service offered by The Globus Alliance at the University of Chicago [6]. It is aimed at the scientific community and, introduced in November 2010, is one of the earliest examples of an MFT service in the cloud. Globus offers fire-and-forget GridFTP file transfers as a service, and provides a web-based front end to their transfer scheduler, as well as a unified interface for requesting authentication credentials for various well-known Grid computing resources. In addition, the interface offers the ability to graphically list and browse directory contents on remote GridFTP servers in real time. Globus also applies a number of heuristic optimizations to its transfers [12] which will be detailed later in the chapter.

Ipswitch's MOVEit Cloud[3] is an MFT service aimed at enterprise organizations with large-scale data requirements [5]. MOVEit Cloud is built on top of Ipswitch's MOVEit File Transfer application. MOVEit Cloud supports a number of protocols and authentication mechanisms, secure person-to-person file transfers, and is HIPAA and PCI compliant.

[1]The Health Insurance Portability and Accountability Act of 1996—a United States legislative act regarding the privacy of medical records.

[2]http://globus.org/.

[3]http://www.moveitmanagedfiletransfer.com/.

Mover[4] is another MFT application designed for use with popular cloud-hosted data storage systems such as Dropbox and SkyDrive, though it also supports transfers via FTP and WebDAV [2]. Mover provides an interface for browsing cloud storage systems and a web-based REST API for interacting with the service programmatically. It can be used to schedule future and recurring data transfers, and offers a simplified method for reusing transfer parameters using transfer templates.

StorkCloud[5] is an MFT application created by the Data Intensive Distributed Computing Lab at the University at Buffalo (Fig. 1). It provides support for a number of data transfer protocols and storage systems, including FTP, GridFTP, HTTP, SMTP, BitTorrent, SCP/SFTP, and iRODS, as well as a collection of protocol-agnostic transfer optimization algorithms. The architecture of StorkCloud will be discussed in detail as a case study of an MFT system in the next section.

4 StorkCloud

StorkCloud is an MFT application based on open source software and is available to the public free of charge. This chapter will take an in depth look at StorkCloud as a case study on the design and implementation of MFT systems.

The major components of the StorkCloud system include:

- an extensible multi-protocol transfer job scheduler for queuing, scheduling, monitoring, and optimizing data transfer jobs;
- a directory listing service (DLS) for prefetching and caching remote directory metadata in the cloud to minimize response time to users;
- a web API adhering to representational state transfer (REST) design principles;
- pluggable transfer modules which can be used to communicate and negotiate with different data transfer protocols and storage systems; and
- pluggable protocol-agnostic optimization modules which can be used to dynamically optimize various transfer settings to improve performance.

StorkCloud schedules, optimizes, and monitors data transfer requests from users through its lightweight thin client utilities (including an Android application, a web browser interface, and command line tools). The API it exposes through its client interface layer can be used by third-party clients and libraries, allowing for StorkCloud to be used as a data connectivity layer in federated cloud systems. The StorkCloud core is written in Java and is open source. The source code can be downloaded from the Stork GitHub repository.[6]

[4]https://mover.io/.

[5]https://storkcloud.org/.

[6]https://github.com/didclab/stork.

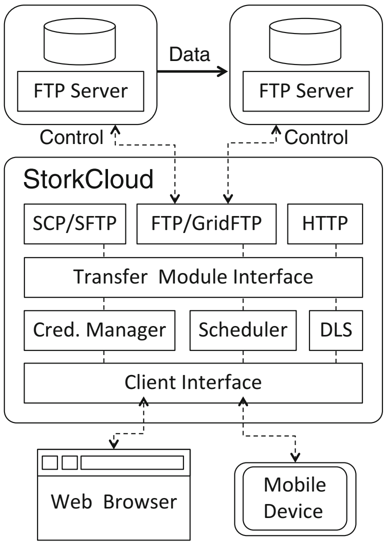

Fig. 1 This illustration depicts the interactions between StorkCloud system components

4.1 StorkCloud Scheduler

StorkCloud's scheduler is a modular, multi-protocol task scheduler which handles the queuing and execution of data transfer jobs and ensures that they complete successfully and in a timely manner. The scheduler's external module interface allows arbitrary protocol support to be implemented as standalone modules and introduced to the system with minimal hassle. As the core component of the StorkCloud system, the scheduler's job is to take in transfer jobs and provide clients with information about the progress of jobs upon request.

Jobs submitted to the scheduler are assigned a numerical identifier—the job ID—which can be used to reference jobs in subsequent requests. The scheduler can be queried to obtain a job status report, which includes information such as the source and destination endpoint URLs, the job state (e.g., scheduled, in progress, failed, complete), the size of the transfer in bytes, the progress of the transfer, instantaneous and average transfer speeds, job submission and start times, and estimated transfer completion time.

StorkCloud provides additional reliability to cloud data transfers via data transfer checkpointing and checksumming for protocols that support them, as well as alternative protocol fallback mechanisms. These are especially useful in large file transfers where the likelihood of errors over the lifetime of a transfer is increased.

StorkCloud also provides mechanisms to monitor end-to-end data transfer tasks to provide clients with real-time progress information as well as to detect failures and performance problems as early as possible. StorkCloud's error reporting framework can distinguish the locus of failure (e.g, network, server, client, software, hardware) in the event of problems, classify problems as transient or permanent, and provide possible recovery options. These error detection, classification, and recovery mechanisms provide greater reliability and agility to transfers performed by the system.

The StorkCloud scheduler is based heavily on the Stork Data Scheduler [16]. The Stork Data Scheduler is considered to be one of the first examples of data scheduling and optimization tools and has been actively used in many data-intensive application areas including coastal hazard prediction and storm surge modeling, oil flow and reservoir uncertainty analysis, numerical relativity and black hole collisions, digital sky imaging educational video processing and behavioral assessment, and multiscale computational fluid dynamics.

4.2 Directory Listing Service (DLS)

StorkCloud's Directory Listing Service (DLS) provides a metadata retrieval service to clients to enable efficient remote file system browsing before issuing a data transfer request. Conceptually, DLS is an intermediate layer between StorkCloud thin clients and arbitrary remote data storage systems that provides access to directory listings as well as other metadata information in a unified format. In that sense, DLS acts as a centralized metadata server hosted in the cloud. When a thin client wants to list a directory or access file metadata on a remote server, it sends a request containing necessary information (i.e., URL of the top directory to perform listing on, along with required credentials) to DLS, and DLS responds back to the client with the requested metadata.

During this process, DLS first checks if the requested metadata is available in its cache. If it is available in the cache (and the provided credentials match the associated cached credentials, and the cache entry has not expired, etc.), DLS directly sends the cached information to the client without connecting to the remote server. Otherwise, it connects to the remote server, retrieves the requested metadata, and sends it to the client. Meanwhile, several levels of subdirectories will be prefetched in the background and cached under the assumption that the user will visit one of the subdirectories in the near future [24].

Any metadata information handled by DLS will be cached and periodically checked with the remote server to ensure freshness of the information. Clients also have the option to refresh/update the DLS cache on demand to bypass the cached metadata and make sure they are receiving the most up-to-date directory listings and metadata.

4.3 Web API and Thin Client GUIs

StorkCloud exposes a RESTful web API that allows thin clients—or even other cloud services and applications—to log in to the system, schedule and control transfer jobs, perform remote directory listings, manage user credentials, and more. Responses to REST requests are represented in JSON, allowing for easy development of browser-based thin client applications. The web API can also be used to develop hybrid web applications which may use StorkCloud's data transfer or metadata retrieval services in conjunction with other cloud-based services.

StorkCloud provides two thin client user interfaces: a web browser interface accessible through the StorkCloud website and a native Android client. Through these interfaces, users can visually observe transfer progress in real time and stop, pause, and cancel transfer jobs using a point-and-click interface. Users can also browse two remote servers simultaneously through a graphical interface which communicates with StorkCloud's DLS to access remote directory contents. Users can traverse remote file systems, select files and directories for transfer, and initiate a transfer between them.

The thin clients provided by StorkCloud cache and prefetch remote directory data provided by StorkCloud's DLS—much in the same way as DLS itself—to provide a much more responsive and interactive user experience.

In addition to these client interfaces which communicate with StorkCloud using the web API, the open source scheduler component comes bundled with a command line utility for communicating with the scheduler directly using either HTTP or a raw TCP connection.

4.4 Transfer Module Interface

StorkCloud acts as a negotiating system between different data storage systems and protocols. In order to do this, StorkCloud must be able to "speak" the protocols of the remote systems it aims to coordinate between. This is done using pluggable, independent "transfer modules" that provide StorkCloud with a uniform interface to a given protocol or storage system.

The transfer module interface allows StorkCloud users to develop modules to support their favorite storage systems, protocols, or middleware easily. Modules can be written in any language recognized by the operating system, as all communication between the transfer modules and scheduler is done in JSON. Users who want to have tighter integration with the system as well as better communication performance may implement transfer modules in Java to communicate directly with the StorkCloud scheduler in memory.

StorkCloud supports a mechanism for protocol translation for cross-protocol data movement using the StorkCloud system as a rendezvous point. It also offers direct access to file data through its HTTP interface, allowing other StorkCloud thin clients and third-party applications to access data though any supported protocol or storage system, with StorkCloud operating as a proxy.

4.5 Optimization Modules

StorkCloud can perform protocol-agnostic optimization of data transfers using pluggable optimization algorithms. Optimization modules (also called optimizers), similar to transfer modules, can be plugged into the server, and incoming jobs can then request an optimization algorithm to be used for the transfer. Optimizers advertise which parameters they are designed to optimize, and transfer modules can likewise advertise which parameters they allow to be adjusted.

If a transfer module allows an optimization algorithm to be used, it queries the optimizer for sample parameters, runs a sample, and reports the throughput back to the optimizer. The optimizer uses the reported information to determine parameters for the next sampling, and continues until either the transfer is complete or the sampling phase is over. This design allows optimizers to be protocol-agnostic—as long as the transfer module supports the features the optimizer exposes, neither needs to know the other's implementation details.

StorkCloud implements a number of dynamic optimization techniques as optimization modules to provide a method for determining which combination of parameters is "just right" for a given transfer. The optimization techniques StorkCloud implements try to maximize transfer throughput by choosing optimal parallelism, concurrency, and pipelining levels through combinations of sampling, file set analysis, heuristic clustering, and learning algorithms applied to historical transfer statistics.

The optimization algorithms StorkCloud supports will be discussed in the following sections.

5 Transfer Level Throughput Optimization

Oftentimes during the course of a data transfer, one may experience periods of poor transfer performance where transfer throughput drops to mere fractions of maximum possible network capacity. Sometimes such effects are intermittent and/or out of the control of the user, such as during times of heavy network utilization on shared networks. However, poor transfer performance can be due to a number of other confounding factors, e.g., underutilization of end system CPU cores, low disk I/O speeds, traffic at inter-system routing nodes, unsuitable system-level tuning of networking protocols, servers not taking advantage of parallel I/O opportunities. Many of these effects can be remedied by properly configuring application-level transfer settings at either the source or destination endpoints and dynamically applying combinations of optimization techniques.

This section will cover algorithms and methodologies for optimizing data placement operations from the perspective of an MFT application.

5.1 Optimization Techniques

Per-transfer optimizations can be used to increase the goodput[7] of an individual data transfer. Adjusting transfer parameters and applying different techniques can play a significant role in increasing transfer throughput. However, determining the appropriate transfer settings and the degree to which techniques should be applied can be difficult, and poor application of the techniques can either cause underutilization of the network or overburden the network and degrade the performance due to factors such as increased packet loss. Inappropriate application of certain techniques can also violate network policies or cause service disruption in environments with shared resources. It is therefore important that care is taken when applying optimization techniques so as to avoid potential issues (Fig. 2).

A number of transfer options and techniques can be applied to many different file transfer protocols, and the appropriateness of their application differer depending on the nature of the end-system subnets, storage systems, and network interconnects. This section will talk about some of these techniques and parameters from the perspective of an MFT system using the following definitions:

- **Pipelining**—This involves queuing up multiple sending or receiving commands at the end-systems in control channel-based transfer applications, as opposed to waiting for transfer to complete before issuing subsequent commands. This helps mitigate the effect of latency in a multi-file transfer.
- **TCP tuning**—A large majority of data transfer protocols are based on TCP, making TCP tuning techniques a valuable tool for optimizing data transfers. In particular, TCP tuning refers to reconfiguring end system TCP buffer sizes to increase performance on networks with high Bandwidth–Delay Products. However, the effectiveness of this technique only goes so far, as oftentimes end systems enforce a maximum buffer size that is less than optimal, requiring the use of other techniques.
- **Data channel parallelism** (or just **parallelism**)—This refers to the use of multiple aggregated data streams (e.g., TCP connections) to a single endpoint, and can be used to overcome the effect of system level limitations on buffer size. The throughput of the aggregate channel approximates that of a single connection with buffer sizes equal to the sum of the individual stream buffer sizes.
- **Concurrency**—This technique involves transferring different files simultaneously, which can take advantage of concurrent I/O in parallel and distributed storage systems. In some application protocols (e.g., FTP and HTTP), this is achieved using parallel control channel or session connections, and in those cases can be used to almost identical effect as parallelism, even when the underlying storage system does not allow parallel I/O.

[7] **Goodput** is the number of useful bits of information transmitted per unit time in a data transfer, in distinction to the amount of bandwidth actually consumed. The ratio of goodput to throughput is the **transfer efficiency**.

- **Striping**—This is the use of multiple source and/or destination endpoints to transfer file data to or from a shared (usually networked) storage subsystem. Like concurrency, this takes advantage of parallel I/O in the storage system, but the use of multiple endpoint hosts also allows it to take advantage of parallel CPUs and sometimes network routes. In the context of network data transfers, the concept of striping is different from, though analogous to, the concept of striping in a disk storage array.[8]
- **Compression**—This can increase the efficiency of a transfer by increasing the number of useful bits of information transferred per transmission unit. However, the use of compression comes at increased computational overhead at both endpoints, and might not be worth it in cases where file data is highly random and thus of poor compressibility.[9]

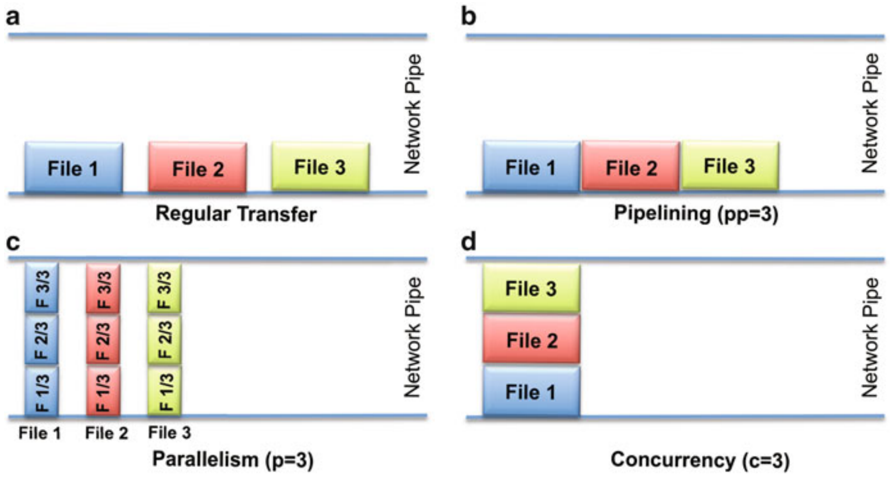

Fig. 2 Effects of pipelining, parallelism, and concurrency on network load

[8]In the context of storage systems, striping refers to dividing file contents across several disks in a RAID to improve read throughput.

[9]Specifically this refers to *lossless* compression, as the file data must be totally reconstructible at the destination endpoint. This chapter does not consider *lossy* compression for purposes of data transfer, though it might be useful depending on the application. Imagining examples of such applications is left as an exercise for the reader.

5.2 Dynamic Optimization

These optimization techniques can be used in combination to different degrees to improve the efficiency of the transfer, insofar as the underlying transfer protocol supports them. However, the degree to which each technique should be used—and when—depends highly on the configuration of the network and end systems, and temporal network conditions. Oftentimes these factors are not explicitly known by users initiating transfers, and so automatic optimization subroutines are an enticing feature for an MFT application to have. Such optimization subroutines can range from simple heuristics that optimize according to file size or historical performance to advanced algorithms that discover network conditions on the fly and tune transfer parameters accordingly.

5.3 Examples in MFT Systems

Globus applies a heuristic optimization for GridFTP transfers based on the average file size of a dataset. At the time of writing, their optimization heuristic always transfers two files concurrently, and chooses parallelism and pipelining levels according to the following rules [12].

- If there are more than 100 files with an average file size smaller than 50 MB, it uses two parallel data channels per file and pipelines up to 20 outstanding commands a time.[10]
- If the average file size is larger than 250 MB, it uses eight parallel data channels per file and pipelines up to five outstanding commands.
- In the default case, it uses four parallel data channels and pipelines up to 10 outstanding commands.

StorkCloud employs a number of dynamic throughput optimization algorithms designed for optimizing different sets of transfer parameters, which users may select when they submit a transfer job. Some of these algorithms "sense" the network between the remote endpoints by performing sample transfers and measuring transfer performance. Others refer to performance information from past transfers between the same endpoints, or use additional information about end-system and configuration to choose theoretically optimal transfer settings. The algorithms provided by StorkCloud include:

[10]In the case of Globus, a pipelined command does not necessarily correspond to one data transfer, meaning pipelining in this sense does not precisely fit the definition given earlier in the chapter. Nevertheless, the relationship between pipelined commands and pipelined data transfers is effectively linear.

- a number of parallel stream modeling and prediction algorithms [14, 22, 23],
- the Parallelism–Concurrency–Pipelining optimizer which uses historical database information and clustering, and
- the Single Chunk Concurrency and Multi Chunk Concurrency algorithms which optimize parallelism, concurrency, and pipelining using clustering and heuristics [8].

Some transfer options can also be used to limit the maximum speed of a data transfer. For example, limiting the TCP buffer size will constrain the number of bits that may be transferred in a given window of time, thus imposing an upper limit on the speed of the transfer. Though this may seem counterproductive, it can be useful in cases where a data transfer is low priority or has a far-off deadline, and minimizing the strain a transfer puts on the network might be desirable. This technique is especially useful when scheduling simultaneous transfers with known start times and deadlines.

6 Scheduling Optimization and Reservation

Aside from optimizing individual data transfers for maximum performance, MFT systems also have the responsibility of coscheduling data transfer jobs of widely variable scale with arbitrary earliest start times and deadlines specified by clients.

As was mentioned in the beginning of the previous section, drops in transfer throughput can sometimes be time-dependent, as is often the case with slowdowns during hours of peak usage. These predictable periods of poor performance can sometimes be mitigated through effective use of timing strategies, especially in cases of very large transfers. Strategies which take transfer priority and desired transfer completion time into account can also increase overall throughput and the number of transfers which complete successfully before their due time.

In practice, an MFT system will be presented with a variety of data transfer tasks with different requirements. This can include small jobs that should complete as quickly as possible, to large, long-running jobs that might have a much broader window of time for completion. Much of the time these transfer jobs are independent and can be coscheduled without the risk of competing for resources (e.g., bandwidth at inter-system routing nodes or end system storage devices). However, there will inevitably be jobs scheduled with destinations that have overlapping routes or are transferring from the same source endpoint, as well as jobs that have a deadline in the far future and need not begin immediately. In these cases, MFT applications are in an advantageous position to make scheduling decisions that reduce resource contention and network load, and maximize the number of deadlines met.

This section will discuss algorithms and practical considerations for performing coscheduling from the perspective of an MFT system.

6.1 Coscheduling Algorithms

In the context of data transfer, **coscheduling** is the process of scheduling multiple concurrent transfer tasks of varying degrees of dependence on shared resources while taking into account time constraints and minimizing the **lateness**—time spent incomplete after the deadline—of a job. In this sense, the coscheduling problem is "merely" a problem of mathematical optimization.

A number of algorithms exist for coscheduling transfer tasks to minimize resource contention, maximize long-term throughput, and reduce the probability of missing transfer deadlines. Depending on the algorithm used, different sets of information regarding the underlying network characteristics and the nature and constraints of the data transfers in question may be necessary.

One intuitive approach to coscheduling involves framing it as a variant of the bin packing problem with two-dimensional "objects" representing transfer jobs being packed into "bins" which represent available bandwidth between a given source and destination. The "volume" of the objects being packed corresponds to the size of the data that must be transferred for a particular job, and the "dimensions" of the objects are the throughput of the job at particular times.

The dimensions of the objects can vary subject to the time constraints of the job, so long as the volume remains the same. This allows for some great variability—some "squishiness"—in the shape of the object (job) being packed into the bin. It is even permissible to "split" the objects into unconnected parts by having periods of time in which the throughput dimension is zero.

By varying the dimensions of object—making them "thinner" or "fatter"—or splitting them, it may be possible to fit more objects in the bin while satisfying all the constraints. This packing corresponds to a schedule which satisfies the constraints of all the jobs being coscheduled. The manipulation of the dimensions of the objects being packed manifests in actual systems as taking measures which might at first seem to be counterproductive, such as throttling the rate of a transfer or intermittently pausing transfers. Despite its counterintuitive nature, the concept of throttling jobs to achieve overall greater across multiple jobs throughput is an established and well-explored technique [19].

One difficulty with the bin packing approach, however, is introduced by the fact that the volume of the bins cannot be so simply defined as "available bandwidth". Indeed, faithfully representing the nature of the Internet in the reduced problem would involve overlapping and interdependent bins corresponding to the convoluted nature of data paths through the Internet. This detail is likely too fine to capture in the metaphor of bin packing. Nevertheless, the bin packing reduction of the coscheduling problem can still be applied with useful results [17].

6.2 Estimation with Historical Performance Data

In addition to having knowledge about transfers in advance, MFT applications can also take advantage of historical transfer performance to make better decisions about future data transfers. This information can be fed into learning algorithms to discover patterns in network usage and make better predictions about optimal transfer settings. This historical data can also be exchanged with third-party scheduling applications or even other MFT services to allow for an even richer transfer history database to be built collaboratively.

Combining information about historical and future transfer jobs can also be used to develop better methods for estimating transfer completion time. This information can be used by MFT schedulers for the sake of making smarter scheduling decisions and for providing users with better estimations of data transfer duration.

Using its historical transfer database and information about ongoing and scheduled transfer jobs, StorkCloud is able to provide information to its clients regarding available end-to-end throughput for the user, the estimated total time it would take to transfer a particular dataset, and the parameters that need to be used in order to achieve the highest end-to-end throughput. If the information necessary to make this predication cannot be found in the historical database, StorkCloud can perform dummy transfers on the fly and use the results to make predictions.

This estimation service allows users to test network resources and conditions, and lets data transfer operations be scheduled in advance with preferred time constraints given by the user—i.e., the requested earliest start time and desired latest completion time. It also lets users and higher level meta-schedulers plan ahead and reserve a time period for their data movement operations. This service can potentially be used to eliminate long delays in transfer completion and increase utilization by giving opportunities to provision required network and storage resources in advance, and also enables third-party data schedulers to make more informed scheduling decisions by organizing requests and focusing on a specific time frame to maximize performance and resource utilization.

6.3 Practical Considerations

Although in some cases information regarding underlying networks may be provided by users, ideally such information would be discovered by the MFT system itself. However, it may not always be possible to do so, especially when conducting third-party transfers as many transfer protocols and applications do not offer ways to collect diagnostics information that may be necessary to determine certain network characteristics.

For example, because of its decentralized nature, data transfers between Internet hosts may not always take the same route. Furthermore, the routes a given transfer may take might not be even discoverable by the MFT system (or sometimes even the remote hosts themselves), because diagnostic protocols for discovering routing information are not universally implemented and much of the time are blocked by intermediate routing nodes for purposes of security. This means it might not always be possible to determine with certainty which data transfers might have overlapping routes, and thus potentially interfere during data transfers.

Even if the route a given data transfer will take is knowable, because of the heterogeneous nature of the Internet and the lack of diagnostic protocols for doing so, it is generally difficult to determine the capacity and transient load of the links in a route. This makes doing beforehand estimations of the maximum data rate of a transfer, even knowing the routes it will take, a difficult task.

Another difficulty arises in actually controlling transfer rate and guaranteeing a transfer will be able to maintain a given rate for the entirety of its scheduled time. The extent to which an MFT application can utilize these techniques varies depending on the capabilities of the underlying network, the system's ability to control and sense the network between two end systems remotely, and the nature of the transfer protocol underlying the transfer. The use of reservations on networks that support them can give an MFT application much more control over and predictive power regarding a transfer.

One other difficulty is that, given that the bin packing approach is NP-hard, it is infeasible to expect that an exactly optimal coscheduling can be found in every case. It is more likely then that a heuristic approach will be necessary in real applications. Combining a heuristic bin packing approach with machine learning techniques applied to historical transfer data can likely produce near-optimal schedulings without succumbing to the potential pitfalls of non-polynomial algorithms.

7 Potential Applications of MFT

Many practical applications exist for MFT in the cloud ecosystem. This section will list a few possible applications in a cloud-connected environment.

7.1 Cloud Data Placement Middleware

One application that was mentioned earlier in the chapter was using MFT as a sort of "glue" to bridge the gap between cloud storage and cloud computation services. Centralized data transfer managers have been used successfully to manage dataset staging in locally distributed computation systems, e.g. HTCondor [16]. MFT would fulfill an analogous role and confer the same benefits for cloud-based computation systems.

An MFT system can be used to stage data between cloud storage systems and cloud computation systems, freeing users from needing to manage staging themselves and also taking advantage of transfer and scheduling optimizations provided by the MFT system.

7.2 Backup Management and Replication

MFT systems can also perform automatic backup of data in cloud storage or replicate and synchronize data across multiple cloud services. Individuals and organizations who make regular backups of data can outsource this task to cloud-hosted MFT systems and take advantage of reliability guarantees and optimized data transfers.

7.3 Data Transfer for Thin Applications

One of the useful properties of cloud service systems is that they give client applications the opportunity to offload expensive or difficult tasks to remote systems. The client then only has to worry about communicating with the remote system and presenting responses from the server to the user.

MFT systems that offers a programmatic interface for transferring data and navigating remote file systems could be integrated into a thin client application to allow it to support multiple protocols and optimization algorithms without increasing the size of the application or burdening the developer to worry about application details. These applications could even be very lightweight web applications that run in the browser, allowing web applications to perform file transfers and access file metadata for multiple protocols and storage systems—something that has until now been difficult to do in web applications.

7.4 Going Further with MFT

The uses of MFT systems can also reach beyond simply managing data transfers and fetching directory listings. Indeed, some creative cases may elicit the need for categorization as something other than “managed file transfer”. Advanced MFT systems are essentially protocol polyglots, and may be extended to take advantage of this.

Consider that MFT systems are necessarily capable of listing and crawling storage systems. Add an indexing mechanism, and it can become something else entirely. Imagine providing a service that allows organizations to index and search their own data stores, and easily locate, collate, and compare organizational documents no matter where or how they are stored. Typically such services are provided only for particular storage systems, but with MFT-as-abstraction layer, the underlying system can be anything. Imagine applying this to build a search engine on an index of publicly accessible storage systems—Google, but for more than just common web protocols.

Consider now that an MFT system that performs protocol translation is essentially a "Swiss Army knife" of storage system clients. Imagine going an extra step and making it a "Swiss Army knife" of server facades as well. Such systems could act as gateway to any supported protocol or storage system, accessible by anything that speaks the "language" of any other supported protocol or storage system. For example, a system that speaks FTP would be able to interface with a system that speaks HTTP—the MFT system acting as a translator—and meanwhile both systems would think they're speaking to a "native speaker" of the protocol they are configured to use.

Imagine a world where data transfer is routinely handled by cloud services. Users would not need to install special software in order to transfer files to or from other users or services in the cloud. The specifics of the underlying protocol or storage system would no longer matter. All of the details of the transfer will be offloaded to the cloud, and when the details change the cloud will adapt and users will never need to know.

It is left up to the reader to imagine other such creative use cases. Certainly the possibilities are great when data can flow so readily in the cloud.

Conclusion

This chapter has taken a look at the challenges faced in moving data into, out of, and around in the cloud. We've seen the issues surrounding data solubility that have arisen as data storage and processing have moved into the cloud, and how these issues can be addressed by MFT services.

We've examined the features of an ideal MFT system, and seen how MFT services in the cloud can take advantage of their centralized nature in order to offer benefits over other transfer management solutions. We've seen a number of established examples of research and commercial MFT systems, and we've taken a close look at the architecture of one such system—StorkCloud.

Hopefully this chapter has made a convincing case for MFT. Perhaps in the future MFT will be as commonplace in the cloud as storage and processing are today.

References

1. ARRA/ANI testbed. https://sites.google.com/a/lbl.gov/ani-100g-network.
2. Backup, copy, and migrate files between cloud storage services | Mover. https://mover.io/.
3. Energy Sciences Network (ESnet). http://www.es.net/.
4. Internet2. http://www.internet2.edu/.
5. Ipswitch MOVEit Managed File Transfer. http://www.moveitmanagedfiletransfer.com/.
6. ALLEN, B., BRESNAHAN, J., CHILDERS, L., FOSTER, I., KANDASWAMY, G., KETTIMUTHU, R., KORDAS, J., LINK, M., MARTIN, S., PICKETT, K., AND TUECKE, S. Software as a service for data scientists. *Communications of the ACM 55:2* (2012), 81–88.
7. ALTSCHUL, S. F., GISH, W., MILLER, W., MYERS, E. W., AND LIPMAN, D. J. Basic Local Alignment Search Tool. *Journal of Molecular Biology 3*, 215 (October 1990), 403–410.
8. ARSLAN, E., ROSS, B., AND KOSAR, T. Dynamic protocol tuning algorithms for high performance data transfers. In *Euro-Par* (2013), F. Wolf, B. Mohr, and D. an Mey, Eds., Lecture Notes in Computer Science, Springer, pp. 725–736.
9. CEYHAN, E., AND KOSAR, T. Large scale data management in sensor networking applications. In *In Proceedings of Secure Cyberspace Workshop* (Shreveport, LA, November 2007).
10. CHO, B., AND GUPTA, I. Budget-constrained bulk data transfer via internet and shipping networks. In *The 8th International Conference on Autonomic Computing (ICAC)* (2011).
11. HEY, T., AND TREFETHEN, A. The data deluge: An e-Science perspective. In *In Grid Computing - Making the Global Infrastructure a Reality*, pp. chapter 36, pp. 809–824. Wiley and Sons, 2003.
12. JUNG, E.-S., KETTIMUTHU, R., AND VISHWANATH, V. Toward optimizing disk-to-disk transfer on 100G networks.
13. KIEHL, J., HACK, J. J., BONAN, G. B., BOVILLE, B. A., WILLIAMSON, D. L., AND RASCH, P. J. The national center for atmospheric research community climate model: Ccm3. *Journal of Climate 11:6* (1998), 1131–1149.
14. KIM, J., YILDIRIM, E., AND KOSAR, T. A highly-accurate and low-overhead prediction model for transfer throughput optimization. In *Proceedings of ACM SC'12 DISCS Workshop* (2012).
15. KLEIN, R. J. T., NICHOLLS, R. J., AND THOMALLA, F. Resilience to natural hazards: How useful is this concept? *Global Environmental Change Part B: Environmental Hazards 5*, 1–2 (2003), 35–45.
16. KOSAR, T., BALMAN, M., YILDIRIM, E., KULASEKARAN, S., AND ROSS, B. Stork data scheduler: Mitigating the data bottleneck in e-science. *The Phil. Transactions of the Royal Society A 369(3254–3267)* (2011).
17. LEINBERGER, W., KARYPIS, G., AND KUMAR, V. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Parallel Processing, 1999. Proceedings. 1999 International Conference on* (1999), IEEE, pp. 404–412.
18. LONI. Louisiana Optical Network Initiative (LONI). http://www.loni.org/.
19. SOUDAN, S., CHEN, B. B., AND VICAT-BLANC PRIMET, P. Flow scheduling and endpoint rate control in gridnetworks. *Future Generation Computer Systems 25*, 8 (2009), 904–911.
20. TUMMALA, S., AND KOSAR, T. Data management challenges in coastal applications. *Journal of Coastal Research special Issue No.50* (2007), 1188–1193.
21. XSEDE. Extreme Science and Engineering Discovery Environment. http://www.xsede.org/.
22. YILDIRIM, E., YIN, D., AND KOSAR, T. Prediction of optimal parallelism level in wide area data transfers. *IEEE TPDS 22(12)* (2011).
23. YIN, D., YILDIRIM, E., AND KOSAR, T. A data throughput prediction and optimization service for widely distributed many-task computing. *IEEE TPDS 22(6)* (2011).
24. ZHANG, B., ROSS, B., TRIPATHI, S., BATRA, S., AND KOSAR, T. Network-aware data caching and prefetching for cloud-hosted metadata retrieval. In *Proceedings of the Third International Workshop on Network-Aware Data Management* (2013), ACM, p. 4.