Scientific Data Distribution Network Based on P2P Transmission Technologies

Parunakian D.A. Isaev E.A., Ovchinnikov I.L., Dumsky D.V., Zaytsev A.Yu.

Institute of mathematical problems of biology RAS

Barinova W.O.

Skobeltsyn Institute of Nuclear Physics MSU

Abstract. This work studies the possibility of using a widely adopted peer-to-peer (P2P) data transmission protocol named BitTorrent for the purpose of building a data storage and distribution system that would allow numerous institutions to access vast unstructured volumes of scientific data. We discuss the practical efficiency of using this technology depending on the typical file sizes and readily available software, the ratio of read and write operations, computational power available at the tracker server, the number of mirrors and simultaneously connected clients.

Overview of common scientific data distribution methods

Modern research operations produce vast and still growing amounts of data. For instance, LSST (Large Synoptic Survey Telescope) outputs 20Tb [1] observational data per day, and the four main LHC experiments combined generate over 40Tb per day [2]. Meanwhile, the demand for global collaboration and data access grows as well, since it is becoming exceedingly difficult for a single research group to process and to analyse such amounts of data.

In order to give geographically remote partners possibility to quickly obtain and analyse the data it is necessary to a) store data using common and/or well-documented formats, and b) rationally choose data transfer protocols and mechanisms based on the supposed loads, data characteristics, technical and organisational requirements.

We now will discuss the first question more closely. In some cases where the data is being accessed only by a small consolidated group of researchers it is not uncommon to see proprietary formats specific to the software package used by that group or formats developed by the group or a research area to emerge. However, as the number of participants and the demand for the data grows, a need arises to agree upon a common open data format. There are numerous examples of such formats; we will only discuss the most widely adopted ones, namely, HDF, CDF, FITS NetCDF formats that are often used in astronomy, space science and related fields of study.

HDF

The Hierarchical Data Format (HDF) [3] was originally created by the NCSA; nowadays its development is overseen by the HDF Group. The format, which is already at its first version (incompatible with older ones) is based on the idea of hierarchical structure of multidimensional arrays. This format is actively used in meteorology, near-Earth space research and other Earth and space sciences where storing time-sorted measurements is required. It allows to achieve high read and write performance often better than that of traditional relational database systems [4].

CDF

The CDF format [5] has been developed by NASA. Just as HDF, it stores the data as multidimensional arrays with that important distinction that it supports automatic compression and decompression of the arrays stored. CDF uses two main field types: rVariables and zVariables. All rVariables in a file must have the same length, which is quite inconvenient when dealing with heterogenous types of data in a single file. The zVariables do not have such a limitation, however the application programming interfaces for manipulating them are currently underdeveloped and thus not suitable for production [6]. One should note that the NetCDF format which was originally based on CDF is no longer compatible with CDF but has instead gained compatibility with HDF5.

FITS

FITS [7] (Flexible Image Transport System) is popular with astronomy researchers for storing imagery and similar data. Among its strengths we should note that it has an ASCII metadata header that contains the coordinate system used and other useful information as key-value pairs, which allows a user to easily read the header using any text viewer or a simple programmatic analyser and to determine whether the file satisfies the criteria desired.

This concludes our brief overview of the scientific data storage formats. We now will discuss granting access to the database to remote users. The many solutions of this problem can be roughly split into two categories:
a) storing data in a data center with an attached supercomputer, and granting the users ability to upload custom software to the machine and to execute it on-site;
b) distribute the data and store/process it locally on the users' computers.

In this work we concentrate on optimal implementation of the second method applied to the problem of distributing observational data collected by the Spektr-R spacecraft in the course of the Radioastron experiment.

Overview of data transfer protocols

Traditional data transfer protocols: FTP, HTTP and their modifications

Let's first review the most widely spread protocols of sending data over IP networks, namely FTP and HTTP.

The File Transfer Protocol (FTP) [8] is one of the oldest protocols used on the Internet; the specs on its first version were published in 1971. Its main peculiarity is that it is a multiport protocol: it uses one TCP channel for listening to commands from established clients and one TCP channel to transmit data. FTP supports two modes, active and passive. In the active mode the client application tells the server which port on the client machine it should connect to in order to execute the requested file transfer operation, and in the passive mode the client establishes the appropriate connection itself.

The Hypertext Transfer Protocol (HTTP) [9] is another widely used protocol, owing much of its popularity to the fact that it is an important part of the WWW specification, although lately an alternative protocol SPDY (developed by Google) has begun replacing it [12]. Despite the fact that it wasn't initially designed to effectively stream large volumes of data, its widespread adoption stimulated development of several extensions and ingenious but conformant methods of using of some of its capabilities for improving its suitability for this purpose (e.g. interrupted file download resume, multipart data type etc.). These improvements make it possible to further use HTTP with satisfactory efficiency for transmitting large volumes of data.

It should also be noted that both FTP and HTTP lack any security measures altogether and are vulnerable to a wide spectrum of attack vectors; thus additional encryption or tunneling layers and modified versions of these protocols are applied when transmitting sensitive data or giving access to non-anonymous user accounts.

UDT

The UDT protocol [10], previously known as SABUL (Simple Available Bandwidth Utility Library) is being developed since 2001 and is currently one of the best-performing and stable protocol for transfering data via dedicated channels. The most recent fourth version of the protocol, available since 2009, is also capable of utilizing public network links in non-exclusive mode, albeit with reduced efficiency. The main differences between UDT and the other protocols discussed include the utilization of UDP instead of TCP as the channel-layer protocol, implementation of duplex connections with send/receive buffers at each end, sending ACK and NAK status messages to confirm a packet's arrival or to request its resending by the application-layer protocol, and a non-linear algorithm of manipulating the window size.

GridFTP

Specifically for the purpose of convenient and secure distribution of large volumes of data the Globus collaboration has developed a set of improvements upon FTP called GridFTP [11]. Among the extensions of interest one should mention the complicated security subsystem, automatic TCP connection parameter calibration to maximize performance, file fragment transfer and, last but not least, support for simultaneous download of fragments of a single file from several sources.

Technical description of the proposed data distribution network

The basis for our proposed system differs from the ones we reviewed earlier in being based on the peer-to-peer data distribution model instead of the client-server one. We have selected BitTorrent as our main protocol for transmitting files.

The files received from the spacecraft undergo a preprocessing procedure, after which they are deployed to special storage units on several independent servers. Such an approach allows to reduce the complexity of the load balancing system by using already available data transmission protocols for that. Each of these storage units is a torrent peer; storage units that contain the required file fragments are selected automatically by the newly connected peer based on the data provided by the tracker.

Before the download commences the "client" peer tells the tracker (whose URL is specified in the metadata file) the hash of the file that it attempts to obtain. The tracker replies with a list of addresses of other peers seeding that file; among these will be both the dedicated storage units and the other peers that have already obtained the file or parts thereof. The "client" peer periodically informs the tracker about the process of the file's transmission and receives an updated list of available peers from it; data itself is transmitted without the tracker's participation.

After receiving a file fragment the receiver computes SHA1 hash of the fragment and compares it with the corresponding value stored in the metadata file. If they are equal, the receiver informs known peers that it now has the fragment in question, otherwise it attempts to re-download the fragment.

Since the size of the metafile depends on the number and the size of data file fragments, one must carefully balance the smaller metafile size and the smaller volume of data that has to be re-downloaded in case of a checksum mismatch. Besides, the time required to calculate SHA1 hashes for file fragments has to be taken into account.

Optimizing the system for transmitting very large files

As mentioned earlier, one of the drawbacks of the proposed system is the relatively high computational complexity of metadata file creation and verification of data received, since for every fragment in the file a SHA1 [13][15] hash must be calculated both when computing the metafile and when completely receiving a fragment. Besides, since the hash for every fragment must be stored in the metadata file, decreasing fragment size leads to larger metadata files.

On the other side, using exceedingly large fragments (over 4Mb) may negatively affect data distribution throughput, since each fragment will consist of 256 and more atomic 16kb blocks. This increases the overalll number of incomplete fragments being simultaneously downloaded and reduces the network's efficiency.

In order to speed up transfer of files tens and hundreds of gigabytes large we propose to use middleware software to split scientific data files into several parts of optimal size, as determined by metadata computational efficiency considerations and to concatenate them back on the receiver's computer automatically.

To find the optimal file size we generate files of different sizes with pseudo-random content by reading /dev/urandom, compute the metafile for each such file and record the time spent. We replicate the experiment 25 times and take the averages for each file size. The experiment was conducted on three servers with varying available RAM sizes; the results indicate that as the size of the processed file approaches the size of available RAM we begin observing high MSE values (up to 30% of the average). Apparently this can be explained by the fact that stock BitTorrent metafile generator attempts to load the whole file into memory and has to use the swap partition in order to store it. If we further increase the file size, the efficiency (file size divided by computation time) falls by 75%.

Besides, large MSEs are introduced if the data files are read from an HDD; this error can be fully removed by copying the files to a RAMFS partition for the sake of the experiment. This makes the efficiency nearly constant until the available RAM is exceeded by the file size.

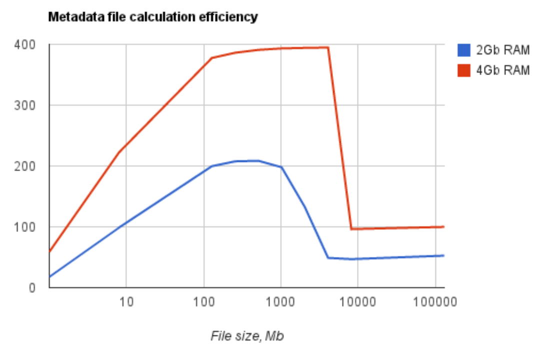

Figure 1. Metafile computation efficiency dependence on the file size. A drastic reduction can be observed at the point where the file size equals the amount of RAM available

Low efficiency with small files is caused by the fact that the time spent on computation per se becomes comparable with the overhead of launching process, Python code interpretation, etc.

We therefore conclude that from the point of view of computational load on the server and taking into account typical hardware specifications of computers used in Russian academia the optimal solution will be to split original files into 1Gb segments and preprocess them using the size of a torrent chunk of 1Mb, which, as show in [14] is a common maximum chunk size.

Client - server scenario

We will now investigate the relative performance of data transmission based on the common protocols discussed above. We are interested in finding out the mean time of transferring a file of the size determined to be optimal. We will conduct two sets of measurements, one for a dedicated link between the initial storage unit and the receiving machine and one for a connection over public networks.

Table 1. Dedicated link

Protocol	Mean throughput	MSE
FTP	882	11
HTTP	884	9
GridFTP	420	8
UDT	490	12
BitTorrent	N/A	N/A

Table 2. Public connection

Protocol	Mean throughput	MSE
FTP	865	81
HTTP	868	74
GridFTP	427	63
UDT	464	11
BitTorrent	263	6

These values are averages over 25 independent runs. We use vsftpd and nginx as FTP and HTTP servers, accordingly and GNU wget as their client. The GridFTP setup was compiled without the use of pthreads. As we can see, using sophisticated protocols doesn't necessarily pay off in simple scenarios like this one and may even negatively impact performance. Besides, according to [17], GridFTP/UDT adoption is reasonable mainly in networks with large RTT values where the effect of using UDT instead of TCP layer allows to increase the throughput by a factor of four. We also should note that while UDT processes use more memory and CPU the benefits of using this protocol are only clearly visible with large MTU values (e.g. 9000) and sizeable buffers, while this setup uses a relatively low MTU of 1500. Low performance of BitTorrent in this scenario is explained by the relatively high SHA1 computation costs. Notice that all the traditional protocols demonstrate high MSEs and, thus, low stability over public networks.

Work in progress

We are currently testing the software written using four Amazon Elastic Cloud m1.small (1.7Gb RAM, 1 dedicated virtual core, fixed network throughput) instances as data storage units and one load balancer t1.micro instance (613Mb RAM, up to 2 virtual cores temporarily assigned, low I/O capabilities). We use OpenTracker project [16] as our BitTorrent tracker. The results are expected to be ready shortly and will be published in a follow-up work, as well as on the Internet.

Notes

1. As shown in [18], injecting fake peers into the pool substantially affects the characteristics of the network, and makes its continued utilization altogether impossible upon reaching a certain threshold. Nevertheless, since the list of users of the proposed system is supposedly known beforehand and is not to be changed very often, this problem can be mitigated by filtering connections from all IP addresses except the ones mentioned in the white list.

2. Previously, a similar system called Biotorrents (Morgan G. I. Langille Jonathan A. Eisen, Genome Center, University of California Davis) [19] was available. Its main purpose was to assist bioengineering groups in data distribution. Among its features were the possibility to group torrent metadata files by the version of the same data file and the automatic distribution of updated file availability by RSS.

3. The popular social network Twitter reduced [20] the time required to propagate updates to its server software to over 1000 servers by the factor of 75 by upgrading to BitTorrent based update delivery system.

References

[1] http://www.lsst.org/lsst/science/concept_data

[2] http://public.web.cern.ch/public/en/LHC/Computing-en.html

[3] http://www.hdfgroup.org/HDF5/doc/PSandPDF/HDF5_UG_r187.pdf

[4] SINP space physics data storage and access system (V. Barinova, V.Kalegaev, D.Parunakian - D.V. Skobeltsyn Institute of Nuclear Physics Lomonosov Moscow State University) PRBEM1-0014-10 \#7084, COSPAR10

[5] http://cdf.gsfc.nasa.gov/

[6] An Introduction to Distributed Visualization (Jan Heijmans, Delft University of Technology Faculty of Information Technology and Systems)

[7] http://fits.gsfc.nasa.gov/standard30/fits_standard30.pdf

[8] http://www.ietf.org/rfc/rfc959.txt

[9] http://www.w3.org/Protocols/rfc2616/rfc2616.html

[10] http://udt.sourceforge.net/doc/draft-gg-udt-03.txt

[11] http://www.ogf.org/documents/GFD.20.pdf

[12] http://mbelshe.github.com/SPDY-Specification/draft-mbelshe-spdy-00.xml

[13] http://wiki.theory.org/index.php/BitTorrentSpecification

[14] http://wiki.vuze.com/w/Torrent_Piece_Size

[15] http://www.ietf.org/rfc/rfc3174.txt

[16] http://erdgeist.org/arts/software/opentracker/

[17] http://gridftp.bio-mirror.net/biomirror/

[18] Kenji Leibnitz, Tobias Hofeld, Naoki Wakamiya, Masayuki Murata.
Peer-to-Peer vs. Client/Server: Reliability and Efciency of a Content
Distribution Service.

[19] http://www.plosone.org/article/info:doi/10.1371/journal.pone.0010071

[20] http://engineering.twitter.com/2010/07/murder-fast-datacenter-code-deploys.html