Scalable Data Collection for Internet-based Digital Government Applications

[Appeared in Proceedings of the 1st National Conference on Digital Government Research, 2001]

W. C. Cheng   C.-F. Chou   L. Golubchik   S. Khuller   H. Samet

Department of Computer Science, University of Maryland, College Park, MD 20742
{william,chengfu,leana,samir,hjs}@cs.umd.edu
http://www.cs.umd.edu/~{william,chengfu,leana,samir,hjs}

Topic Areas: Delivery of public surveys and data gathering.
Delivery of geospatial, textual, audiovisual, statistical, and other information.
Cooperation among federal, state, and local government agencies.

Abstract

Data collection (or *uploading*) is an inherent part of numerous digital government applications. Solutions which have been developed over the years for *download* problems do not apply to uploads. Hence, scalable uploading of data over the Internet is an important, and until now an open, research problem.

1 Data Collection Applications in Digital Government

Government at all levels is a major *collector* and provider of data. There are clear benefits to disseminating and collecting data over the Internet, given its existing large-scale infrastructure and wide-spread reach in commercial, private, and government domains. In this paper, we focus on the *collection of data over the Internet*, and specifically, on the *scalability* issues which arise in the context of Internet-based massive data collection applications. By data collection, we mean applications such as Internal Revenue Service (IRS) applications with respect to electronic submission of income tax forms. Below, we review a few such applications.

Congress wants 80% of tax returns to be filed *electronically* by 2007. Electronic returns are easier to process and contain far fewer errors. Clearly, with (on the order of) 100 million tax returns being filed on April 15th, where each return is on the order of 100 KBytes [5], *scalability* issues are a major concern.

Georeferenced data is subject to both download and upload operations. Downloads are useful when users are sharing data. However, important upload applications also exist. For example, georeferenced data may be gathered by multiple sources, for example personnel in the field taking measurements or automated measurement devices (e.g., using GPS), which must be collected into a single server (or cluster of servers). Another scenario is where a client performs some customized processing on data from one or more data sources and wishes to store it for future use. This processing may be as simple as the selection of data objects satisfying some criteria, e.g., all residential houses within two miles of a river (in database terminology, the result would then represent a materialized view), or may involve complex algorithms and techniques. Thus, in these examples, the client eventually needs to upload the data to the distributed repository for future use. This data can either be results that are only useful for this client/agency (perhaps they are not complete yet), or it can be data that is now available to/shared with other agencies. The scenario of many users uploading at once includes the case of a natural disaster emergency.

The Integrated Justice Information Technology Initiative facilitates information sharing among state, local, and tribal justice components. This program coordinates with a number of other Department of Justice's technology initiatives, including the Global Criminal Justice Information Network Initiative. An integrated (global) information sharing system involves collection, analysis, and dissemination of criminal data. The sizes of the data being collected in this application have a wide variance, for instance they can include text and images. Clearly, in order to facilitate such a system one must provide a *scalable* infrastructure for collection of data.

A number of government agencies support research activities, where the funds are awarded through a grant proposal process, with deadlines imposed on submission dates. For instance, agencies like the National Science Foundation (NSF), which currently uses FastLane, and the National Institute of Health (NIH) have a fairly large number of proposals submitted on regular basis. The entire process involves not only submission of proposals, which can involve fairly large data sizes, but also a review process, a reporting process (after the grant is awarded), and possibly a results dissemination process. All these processes involve a data collection step.

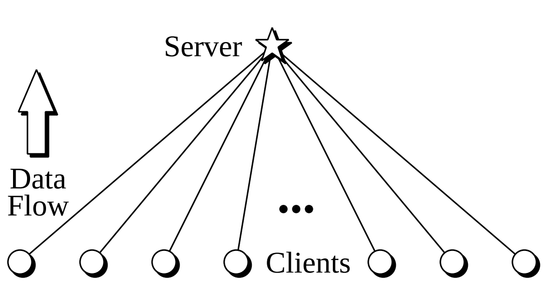
Digital democracy applications, such as online voting during federal, state, or local elections, constitute another set of massive upload applications. These are also deadline-driven applications, with relatively small file sizes, whose results (i.e., outcome of an election) is expected to be computed soon after the uploading deadline. These applications provide opportunities for application-specific data aggregation. Of course, there are numerous other examples of digital government applications with large-scale data collection needs. Throughout this article we will use the IRS example to illustrate our points, mainly due to its familiarity.

2 Working within the Current Internet Technology

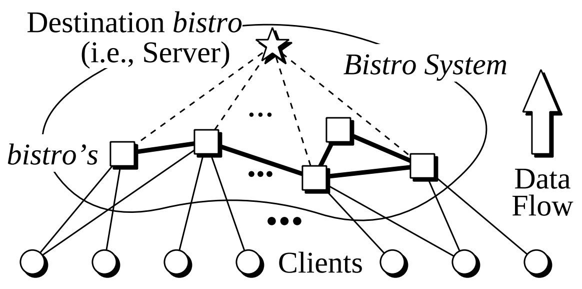
Given the current state of upload applications (i.e., everyone uploads directly to the data's final destination server, as depicted in Figure 1(a)), a specific upload flow, from some client to the destination server, can experience the following potential bottlenecks: (a) poor connectivity of the client, (b) congestion somewhere between the client and the server, (c) overload on the server, or a combination of these bottlenecks. Given these bottlenecks, traditional solutions (or a combination of these solutions), such as buying a bigger server, buying a bigger pipe, and co-locating the server(s) at the ISP(s), exhibit shortcomings including lack of flexibility and lack of scalability.

The poor performance that is likely to be experienced by users of digital government applications, given the existing state of technology, is largely due to how (independent) data transfers work over the Internet, i.e., TCP/IP. That is, TCP/IP is very good at equally sharing the bandwidth between all flows using the same link. To illustrate this point, consider the following example.

Suppose 1 million (out of the possible 100 million) users submit their income tax forms to IRS servers at about the same time. Suppose that IRS places $2,500$ servers around the country, with an average of 400 users uploading to each server simultaneously. And, suppose further that these 400 users share the same *bottleneck* link during the simultaneous uploads to one of these servers (as depicted in Figure 1(a)) while trying to beat the deadline to submit their income tax forms, each receiving $(\frac{1}{400})^{th}$ of the bottleneck link's bandwidth. If in isolation the user would have been able to complete the upload in 30 seconds, then (using back of the envelope calculation) on the *average* a user would have to wait on the order of 2 hours to complete his or her transfer, and in the *worst* case he or she will have to wait on the order of 3 hours.



(a) upload without the *Bistro System*



(b) upload with the *Bistro System* after *Bistro* software is installed on the Server

Figure 1: Uploads with and without the *Bistro* system.

Furthermore, placing $2,500$ servers around the country dedicated to what is essentially a *one day* event is not the most cost-effective approach. (Not to mention that predicting how many are needed this year, and how many more will be needed next year is non-trivial.) Thus, the users suffer due to poor performance, and the IRS suffers due to high costs.

Although TCP/IP is here to stay (at least for the foreseeable future), much can be done at the application layer to circumvent this problem. An application layer solution, in the context of data collection applications, is the focus of this paper.

3 Why Does "Upload" Require New Solutions?

Hotspots are a major obstacle to achieving scalability in the Internet. At the application layer, hotspots are usually caused by either (a) high demand for some data or (b) high demand for a certain service. This high demand for data or services, is typically the result of a *real-life event* involving availability of new data or approaching deadlines; therefore, relief of these hotspots may improve quality of life. For instance, imagine again those 1 million people trying to make the deadline for submitting their tax returns on April 15th, all at 5 minutes to midnight. This would not only flood the IRS servers and result in many people not making the deadline, but it would also create sufficient traffic on the Internet to interfere with many other Internet-based applications that have no relationship with income tax form submissions.

At the application layer, hotspot problems have traditionally been dealt with using some combination of (1) increasing capacity; (2) spreading the load over time, space, or both; and (3) changing the workload. These classes of solutions have been studied mostly in the context of applications using one-to-many, many-to-many[1], and one-to-one types of communication. However, to the best of our knowledge there is no existing work, except ours, on making applications with *many-to-one* communication scalable and efficient. Existing solutions, such as web based submissions, simply use many independent one-to-one transfers. Many-to-one communication corresponds to an important class of applications, whose examples (i.e., *upload* applications) were described in Section 1.

We can view hotspots in most *download* applications as being due to a demand for popular *data objects*.

[1]Although many-to-many data transfers can be achieved using either a set of one-to-many, a set of many-to-one, or both types of data transfers, to the best of our knowledge, existing work on many-to-many data transfer applications is focused on making the one-to-many communication efficient and scalable.

We can view hotspots in most *upload* applications as being due to a demand for a popular *service*, e.g., the income tax submission service, as the actual data being transfered by the various users is distinct. The two main characteristics which make upload applications different from download applications are as follows: (1) in the case of uploads, the real-life event which causes the hotspots often imposes a *hard deadline* on the data transfer service, whereas in the case of downloads, it translates into a desire for low latency data access; and (2) uploads are inherently data *writing* applications while downloads are data reading applications. Traditional solutions aimed at latency reduction for data *reading* applications are (a) data replication (using a variety of techniques such as caching, prefetching, mirroring, etc.) and (b) data replacement (such as sending a low resolution version of the data for image, video, audio downloads). Clearly, these techniques are not applicable in uploads.

Additionally, *confidentiality* of data as well as other security issues are especially important in write-type applications (e.g., in uploading tax forms, papers, and proposals). Another important characteristic of uploads is that, unlike most downloads where data is intended to be consumed immediately upon receipt, uploaded data is often stored at the server for some time before its consumption. We will explain how we exploit this characteristic in the next section.

4 Scalable Framework

In this section we briefly outline the basics of our framework intended as a solution to the scalable data collection problem. A number of important and difficult research problems remain open within this framework, as outlined in [1], as well as within the context of the general problem of data collection over the Internet. Our goal in this article is to illustrate the importance of this, until now neglected, research problem as well as present one possible solution.

Two types of upload applications exist, those with deadlines and those without deadlines. Our framework, termed the *Bistro* system, under development at the University of Maryland [4], can accommodate both types of applications. Below, we discuss our framework in the context of deadline-driven applications with reasonably large data transfer sizes, such as the IRS application described above.

We observe that the existence of hotspots in uploads is largely due to approaching deadlines. The hotspot is exacerbated by the long transfer times. We also observe that what is actually required is an assurance that specific data was submitted before a specific time, and that the transfer of the data needs to be done in a timely fashion, but does *not* have to occur by that deadline because the data is not consumed by the server right away, i.e., IRS agents are not waiting to process all tax forms at one minute past midnight on April 16th.

Thus our approach is to break the deadline-driven upload problem into pieces, (refer to Figure 1(b)). Specifically, we break up our original deadline-driven upload problem into: (a) a real-time *timestamp* subproblem, where we ensure that the data is timestamped and that the data cannot be subsequently tampered with; (b) a low latency *commit* subproblem, where the data goes “somewhere” and the user is assured that the data is safely and securely “on its way” to the server; and (c) a timely *data transfer* subproblem, which can be carefully planned (and coordinated with other uploads) and must go to the original destination. This means that we have taken a traditionally *synchronized client-push* solution and replaced it with a *non-synchronized* solution that uses some combination of *client-push* and *server-pull* approaches. Consequently, we eliminate the hotspots by spreading most of the demand on the server over time; this is accomplished by making the actual data transfers “independent” of the deadline. (Also, note that the only step that must occur before the deadline is the timestamp, i.e., the actual data does not have to be transfered before the deadline.)

For the non-deadline-driven upload problems we can modify subproblem (a) above to provide simple mechanisms (e.g., message digests) which can later be used to verify data integrity.

Note that the above solution is somewhat analogous to sending a certified letter through a postal service with the main difference being that there is an inherent trust in the postal system whereas our solution involves the use of untrusted intermediaries that essentially act as untrusted post offices. Hence *security* and *integrity* of the data become an inherent part of our framework. We refer the interested reader to [3] for one possible security protocol for an IRS-type application.

A Case for Use of Intermediaries
In the Internet, connectivity is usually non-uniform and can be time-dependent. Such connectivity problems contribute to important obstacles to achieving efficient large-scale data collection. Poor connectivity problems are due to either high traffic or low bandwidth connections. Consequently, different hosts experience different connectivity characteristics to the same data source. And hence, an effective solution to connectivity problems over the Internet must by necessity be distributed.

As is clear from past experience with download applications (e.g., Napster-type solutions) great benefits can be obtained through the use of host intermediaries. Furthermore, the above stated solution in the context of the *Bistro* framework also implies the use of intermediaries.

Importance of Resource Sharing
An important characteristic of the *Bistro* framework is the notion of *resource sharing*. It is fairly clear that a more traditional solution of, e.g., buying a bigger cluster for a “one time event” (which may not be “big enough” for the next similar event) is not the most desirable or flexible solution to the upload problem. The ability to share an infrastructure, such as an infrastructure of intermediaries or, in our case, *bistro*s, between a variety of wide-area digital government data collection applications for a variety of government agencies has clear advantages over the traditional solutions described earlier.

Deployment Issues
Bistro is an application layer platform, and hence (unlike say a network layer protocol) it is easily deployable. Installing a *bistro* is as simple as installing a web server. Our intent for deploying the *Bistro* platform is *not* to rely on adding resources (such as hosts) to the Internet. Rather, we envision that people will want to install *Bistro* on their hosts on the public Internet and contribute their resources to the overall *Bistro* infrastructure because it will improve their performance as well. In turn, the existing *bistro*s will discover the new installations and integrate them into a *Bistro* infrastructure. Thus, our architecture will take advantage of *existing* resources and utilize them to their full potential for each upload application.

5 Conclusions

Scalable Internet-based data collection problem has been, until this point, a neglected research topic. The *Bistro* framework has been designed to address scalability problems in Internet-based upload applications over a wide range of applications and problem sizes. In designing, implementing, and deploying *Bistro*, we are gaining knowledge and experience that are fundamental to making large-scale data collection and dissemination, for a variety of applications, a reality.

An attractive characteristic of the *Bistro* framework, with respect to digital government needs, is that it can be gradually deployed and experimented with over the public Internet, simply by installing the *Bistro* system on intermediaries (which can be public or private). Each application (within each agency) can have its own scalability, security, fault tolerance, and other upload characteristics, and these applications and agencies can still share available resources, if so desired, across all *Bistro* servers, with minimal interference. For example, our security protocol (refer to [3]) does not rely on a particular cryptographic system in order to provide the needed privacy and integrity characteristics; hence, each application and each agency can choose one that satisfies its requirements. Efficient and cost-effective sharing of resources between multiple upload applications is an important part of our work.

To date, we have designed the *Bistro* framework [1] and conducted a performance study which demonstrated the potential performance gains of this framework as well as provided insight into the general upload problem [2]. Since confidentiality of data as well as other security issues are especially important in upload applications and in our solution where we introduced untrusted (public) intermediaries (i.e., *bistro*s), we also developed a secure data transfer protocol within the *Bistro* framework which not only ensures the privacy and integrity of the data but also takes scalability considerations into account [3]. However, a great number of open research problems still remain with the *Bistro* framework as outlined in [1]; they are the topic of our ongoing efforts.

References

[1] S. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: a platform for building scalable wide-area upload applications. *Performance Evaluation Review (also presented at the Workshop on Performance and Architecture of Web Servers (PAWS) in June 2000)*, 28(2):29–35, September 2000.

[2] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A performance study of bistro, a scalable wide-area upload architecture. *Submitted for publication*.

[3] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A secure and scalable wide-area upload service architecture. *To appear in the Proceedings of the 2nd International Conference on Internet Computing*, 2001.

[4] http://www.cs.umd.edu/projects/icl/bistro/. *Bistro Project Home Page*.

[5] http://www.irs.ustreas.gov/prod/forms_pubs/fillin.html. *IRS Fill-in Forms*.