Bistro: a Framework for Building Scalable Wide-Area Upload Applications

Abstract
Hot spots are a major obstacle to achieving scalability in the Internet. At the application layer, hot spots 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 hot spots may improve quality of life. At the application layer, hot spot 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.

We note that the classes of solutions stated above have been studied mostly in the context of applications using the following types of communication (a) one-to-many, (b) many-to-many, and (c) one-to-one. However, to the best of our knowledge there is no existing work on making applications using many-to-one communication scalable and efficient (existing solutions, such as web based submissions, simply use many independent one-to-one transfers). This corresponds to an important class of applications, whose examples include the various upload applications such as submission of income tax forms, conference paper submission, proposal submission through the NSF FastLane system, homework and project submissions in distance education, voting in digital democracy applications, voting in interactive television, and many more.

Consequently, the main focus of this paper is scalable infrastructure design for relief of hot spots in wide-area upload applications.

The main contributions of this paper are as follows. We state (a) a new problem, specifically, the many-to-one communication, or upload, problem as well as (b) the (currently) fundamental obstacles to building scalable wide-area upload applications. We also propose a general framework, which we term the Bistro system, for a class of solutions to the upload problem. In addition, we suggest a number of open research problems, within this framework, throughout the paper.

1 Introduction
Hot spots are a major obstacle to achieving scalability in the Internet. At the application layer, hot spots 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 hot spots may improve quality of life. At the application layer, hot spot 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. Some examples of these are data replication (e.g., web caching [6], ftp mirroring), data replacement (e.g., multi-resolution images, audio, video), service replication (e.g., DNS lookup, Network Time Protocol), and server push (e.g., news download, software distribution).

We note that the classes of solutions stated above have been studied mostly in the context of applications using the following types of communication (a) one-to-many (data travels primarily from a server to multiple clients, e.g., web download, software distribution, and video-on-demand); (b) many-to-many (data travels between multiple clients, through either a centralized or a distributed server, e.g., chat rooms and video conferencing); and (c) one-to-one (data travels between two clients, e.g., e-mail and e-talk). However, to the best of our knowledge there is no existing work on making many-to-one communication scalable and efficient (existing solutions, such as web based submissions, simply use many independent one-to-one transfers). This corresponds to an important class of applications, whose examples include the various upload applications such as submission of income tax forms, conference paper submission, proposal submission through the NSF FastLane system, homework and project submissions in distance education [12], voting in digital democracy applications [14], voting in interactive television [10], and many more. Consequently, the main focus of this paper is scalable infrastructure design for relief of hot spots in wide-area upload applications. (In the remainder of the paper we will refer to many-to-one data transfer as “upload” and one-to-many data transfer as “download”.)

1.1 Why Does “Upload” Require New Solutions?
We can view hot spots in most download applications as being due to a demand for popular data objects. We can view hot spots 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...
transferred by the various users is distinct. The two main
c 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 hot
spots often imposes a hard deadline on the data trans-
sfer service, whereas in the case of downloads, it trans-
lates 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 ap-
plications (e.g., in uploading tax forms, papers, and pro-

dals). 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 con-
sumption. We will explain how we exploit this charac-
teristic in the next section. We also note that, 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 fo-
cused on making the one-to-many communication effi-
cient and scalable.

1.2 Our Solution
We observe that the existence of hot spots in uploads is
largely due to approaching deadlines. The hot spot is
exacerbated by the long transfer times. We also observe
that what is actually required is an assurance that spe-
cific data was submitted before a specific time, and that
the transfer of the data needs to be done in a timely fash-
ion. (Note that, this is somewhat analogous to sending
a certified letter through a postal service.) 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
hot spots by spreading most of the demand on the server
over time; this is accomplished by making the actual
data transfers “independent” of the deadline.

1.3 Our Contributions
Our main contributions here are as follows. We:

1. state the problem: to the best of our knowledge this
work is the first effort that considers the many-to-
one communication, or upload, problem and states
the (currently) fundamental obstacles to building
scalable wide-area upload applications;

2. propose a general framework and an architec-
ture for a class of solutions: to construct a gen-
eral architecture for implementing upload applica-
tions, we propose Bistro, an application-layer
framework, which implements a set of primitive services,
such as timestamping, security, commit, and data transfer;

3. discuss the problem of low latency commit: as
stated above, we break the upload problem into
two pieces, corresponding to (a) a real-time time-
tamp, (b) a low latency commit, and (c) a timely
data transfer — in this paper we discuss the low la-
tency commit problem (refer to [1] for details of a

corresponding quantitative study where we charac-
terize potential performance gains and present cor-
responding insight into the upload problem; this
study is not included here due to lack of space);

4. suggest open problems: the solution to the gen-
eral upload problem is non-trivial and cuts across
a variety of areas, including networking, perfor-
mance evaluation, scheduling, load balancing, se-
curity, fault tolerance, and many more; throughout
the paper we suggest a number of open problems,
within the Bistro framework, which must still be
solved in order to build scalable wide-area upload
applications.

2 Related Work
Work related to wide-area data transfers falls into mul-
tiple categories and requires solutions in areas such as
communication networks, performance evaluation, al-
gorithms for load balancing, scheduling and facility lo-
cation, and security. However, to the best of our knowl-
edge none address the problem of making wide-area up-
loads scalable and efficient.

As will become evident in Section 3 our basic ap-
proach (in addition to the breakdown into subproblems)

involves partial replication of services. Replication is
a relatively traditional solution to scalability problems
in Internet-related services, and hence the problems we
consider in this work of assignment of clients to servers,
location of services, and so on (as described in more
detail in Section 3) arise in other applications as well.
(Some of the differences are discussed in Section 4.2.3.) For instance, in [3], the authors consider dynamic server selection in the context of web download applications. Moreover, relief of hot-spots in the Internet through the use of data and service replication (e.g., proxy servers) has been studied extensively in the context of download applications, for instance, as in [7].

In addition, much work has been done on facility location problems [5, 11] in Operations Research and Computer Science. Here, the objective is to open a subset of facilities from a given set \( F \), and to then minimize the sum of distances from a given set of customers to their closest facility. In some formulations the total number of open facilities is constrained, in other formulations a combined objective of facilities and distance costs is minimized. This problem is NP-complete and several good polynomial time approximation algorithms are available for it. However, these solutions do not directly address the problem we are interested in – there are two main differences. The first one is that there is an underlying network and different clients are competing for resources in the network. The second main difference is that we cannot choose a route to the server at the application layer (we can only choose a server and the network chooses the route for us, at least in the Internet). This constraint makes it hard to apply network flow based methods for assignment of clients to servers [8]. These new constraints lead to very interesting optimization problems, as stated in Section 3.

3 Our Upload Framework

In this section we briefly describe our basic framework and its advantages as well as some of the related performance, security, and deployment issues.

3.1 Basic Framework

As stated in Section 1, our approach is to break the upload problem into the following subproblems: (a) timestamp, (b) commit, and (c) transfer, and then design and develop the Bistro architecture which implements solutions to these subproblems using a set of primitive services – which primitive services are used to build an upload application is a function of that application’s requirements.

Given the breakup into subproblems, the original data transfer is now done using two data transfers (1) from a client to one or possibly more hosts on the Internet, which we will refer to as a bistro’s, and then (2) from one or more bistro’s to the server. This flow of data is illustrated in Figure 1. Although Figure 1 only depicts a single upload, it is understood that the bistro’s may be shared by many simultaneous upload activities/applications, each with different deadlines, characteristics, and requirements. Coordination of multiple simultaneous upload applications is an open research problem.

Note that, the client-to-bistro data transfer also produces the timestamp and the commit, i.e., the data is timestamped so the server has a guarantee that it cannot be tampered with after the deadline, and the client receives a commit, i.e., a “receipt” that guarantees that the data will be delivered to the server and that its integrity and privacy will be preserved.

As already stated in Section 1 the timestamp has to be produced before the deadline; the commit has to be performed with low latency, and the data transfer from a bistro to the server has to be done in a timely manner. The exact constraints on all these operations are again a function of the requirements of the particular upload application.

3.1.1 Advantages of the Bistro Framework

We now outline the potential performance problems and considerations, given the current state of upload applications, i.e., everyone uploads directly to the final destination server (refer to Figure 1(a)), in order to illustrate the advantages of our proposed framework (as described above).

In the current state of upload applications, a specific upload flow, from some client to the destination server, can experience the following potential bottlenecks (or hot spots):

1. poor connectivity of the client: some link between that client and the final destination is the bottleneck of the upload process (including the link that connects the client to the Internet);

2. overload on the server link: the link that connects the server to the Internet is overloaded due to too many simultaneous uploads to that server, and this link is the bottleneck of the upload process;

3. overload on the server: the server itself is overloaded due to too many simultaneous uploads to that server, and the server is the bottleneck of the upload process.

Given these bottlenecks, there are several traditional solutions (or a combination of these solutions) that one could consider:

- **get a bigger server**: for example, buy a bigger cluster of workstations to act as the upload server, which is intended to address problem (3) above;

- **buy a bigger pipe**: that is improve the server’s connectivity to the Internet, which is intended to address problem (2) above;

- **co-locate the server(s) at the ISP(s)**: make arrangements directly with the ISP’s to provide upload service at their locations, which is intended to solve problems (1) and (2) above (as well as problem (3) if this service is replicated at multiple ISP’s).

These solutions have a number of shortcomings, including lack of flexibility and lack of scalability. We note that an important characteristic of the Bistro framework is the notion of resource sharing. It is fairly clear that, for instance, 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 infras-
tructure, such as an infrastructure of proxies or bistro’s,
between a variety of wide-area applications has clear ad-
vantages, some of which are outlined below.

Consequently, we believe that Bistro, as outlined
in Section 1 and described in more detail above and in
Section 4, is a better solution than the more traditional
solutions described above, for the following reasons:

- it is more dynamic and therefore more adaptive to
  system and network conditions;
- it provides for more resource sharing opportunities
  and thus can result in a more cost effective solution
to wide-area upload problems; and
- it does not rely on the existence of a private in-
  frastructure (such as the co-location approach
described above), but it does not preclude it either.

In summary, we believe that the Bistro framework is a
more flexible solution that takes advantage of whatever
resources are available in the system and the Internet to
the “best” extent possible.

3.2 Research Issues

In this section we describe a set of problems that need
to be solved in order to design and develop the Bistro
framework proposed above. In this discussion, where
appropriate, we point out open research problems.

3.2.1 Resource Location and Discovery

One important open problem is the location and discov-
er of existing resources (bistro’s) — that is, given a
dynamic infrastructure of bistro’s, for each instance of
an upload application, an important consideration is the
mechanisms and policies used to discover which bistro’s
are available for this upload. This includes how to dis-
cover newly installed bistro’s and how to estimate their
performance characteristics, both, in terms of host and
network capacities in order to make run-time choices of
which bistro’s to utilize.

3.2.2 Placement and Assignment

Another important open problem is assignment of
clients to bistro’s for an instance of an upload applica-
tion. (We briefly describe this problem here and define
it more formally in Section 4.)

More specifically, this involves design of mechani-
isms and policies for determining which bistro should
participate in a given upload and which client should
transfer its data to which bistro, i.e., assignment of
clients to bistro’s. In general, as described in more de-
tail in Section 4, it is not desirable to utilize all existing
bistro’s for an instance of an upload application, even
though this might provide the quickest way of moving
data from the users to the bistro’s (for instance, due to
the overhead caused by managing all the bistro’s and the
overhead of performing bistro-to-server transfer.)

In general, a distributed facility location type
scheme will be required for deciding which bistro
should participate in each upload as well as for assign-
ment of clients to bistro’s. This is a difficult problem
as it also involves prediction of performance of a fu-
ture data transfer (e.g., response time or throughput)
between a specific client-bistro pair based on current net-
work conditions. Our results to date (given in Section 4
and in [1]) indicate that such predictions are non-trivial
and result in difficult research problems.

3.2.3 Security

Adding intermediaries (i.e., bistro’s) in the data trans-
fer has obvious security implications: clearly, it should
not be possible for bistro’s to corrupt the data in tran-
sit in any way. In general, the set of security properties
desirable for an upload service is as follows:

- Integrity: The data cannot be changed in transit by
  any principal.
- Privacy: For some transfers, it may be necessary
to ensure that the data is encrypted and cannot be
interpreted by intermediaries on the transfer path.
- Authentication and non-repudiation: Since the des-
tination now receives data from nodes that are not
the source of the data, it may be necessary to au-
thenticate the source of the data. The mechanisms
employed to authenticate the data should also be
able to discriminate “replays” by malicious bistro’s
and provide non-repudiable transfer.

All of these properties are, in fact, desirable for any data
transfer and many cryptographic techniques have been
developed for implementing these properties [9, 13]. Usually, these techniques assume a powerful adversary capable of intercepting and changing messages in transit. This model is immediately applicable to the Bistro framework, with malicious bistro’s being the adversaries. Thus, we can use existing cryptographic techniques to implement all these security properties for Bistro transfers. The details of our initial security protocol can be found in [4].

3.3 Deployment Issues
Our intent for deploying the Bistro platform is not to rely on adding resources (such as hosts) to the Internet\(^2\). 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.

We first plan to deploy Bistro at academic sites since the resulting performance improvements of our applications, such as paper and proposal submissions, may greatly improve the quality of life for the researchers at these institutions, as the deadlines approach.

4 “Commit” Problem
As already stated in Section 1, our basic approach is to break up the upload problem into: (a) a real-time timestamp subproblem, (b) a low latency commit subproblem, and (c) a timely data transfer subproblem. Furthermore, in Section 3, we stated that (a) and (c) are outside the scope of this paper. In the remainder of this section we focus on the commit problem.

4.1 Extreme Cases
Before discussing our solution, we describe some extreme cases of the framework outlined in Section 3, in order to illustrate the range of performance considerations. These extreme cases are as follows:

1. the final destination of the data transfer is the only bistro (this is essentially the current state of things); or
2. “all” hosts are bistro’s, by this we mean that each client serves as its own bistro, with essentially zero transfer time — it is not clear whether this is “practical”, but regardless, the commit problem is trivial in that case (although the timestamp and data transfer problems are not trivial but are outside the scope of this paper), and hence we do not consider this extreme any further here;
3. each “organization” with an upload client has its own “local” bistro, where granularity of organization is the same as for news (NNTP) servers, DNS servers, and so on. This local bistro can be behind organizational firewalls and may not be accessible from the outside world. In this case, a submission still has to travel to (be pushed) outside of the organization into the public Internet so that the data transfer part of the upload process can take over. Therefore, the commit problem still exists. As far as a user is concerned, the submission is completed as soon as the local bistro issues a receipt (i.e., commit) for the submission. However, the Bistro system will only consider the commit to be completed when the submitted data has traveled to a public bistro\(^3\). Thus, in this case, the commit is broken down into two steps, where the first step is almost as trivial as the one in extreme case 2 above.

4.2 The Middle Ground
It is often difficult to deploy a server infrastructure over the public Internet. In order for the Bistro system to succeed, we must demonstrate that a system with a limited number of bistro’s can provide benefits to the users of this system. In [1] we provide evidence, through a simulation study, that even a system with a limited number of bistro’s can provide such benefits (we omit this study here due to lack of space). In the remainder of this section we discuss the hardness of the commit step.

In our framework (as described in Section 3), there are two basic problems within the commit step that we must consider, which are:

1. assignment problem: this is the situation where the locations of bistro’s are fixed, and the difficulty is in assigning clients to the bistro’s, i.e., deciding which client should upload to which bistro; and
2. placement or selection (plus assignment) problem: this is the situation where the locations of bistro’s are flexible, and hence the difficulty is in both, deciding where to place the bistro’s (i.e., which nodes to choose to act as bistro’s) and how to assign clients to them (note that the placement of bistro’s and assignment of clients to them are not independent problems); from here on, we refer to this as the placement problem.

Below, we define and characterize the assignment and placement problems more formally. However, first we motivate the need to consider the assignment problem, i.e., the problem where not all potential bistro sites are used. That is, given \(N\) potential bistro sites, we only consider placement of bistro’s at \(M < N\) of those sites. This motivation is as follows: (1) as our performance evaluation results in [1] indicate, \(M\) does not have to be very large to obtain most of the benefits of the Bistro architecture, (2) if we take the last step of the upload into consideration, i.e., the timely data transfer to the final destination, then intuitively coordination of transfers from a large set of bistro’s is not necessarily better

\(^2\)Note that deployment over private networks can also be done but is relatively straightforward from the deployment point of view.

\(^3\)If the Bistro system is successfully deployed, we expect bistro installations to be as common as news and mail server installations.
than from a smaller set, and (3) our long-term goal is to allow multiple simultaneous upload applications to use the same Bistro infrastructure, in which case, due to contention, it may be beneficial to allow each application to use a subset of available bistros’s.

Lastly, we note that even the assignment problem alone is hard under general network topologies, as we describe more formally next, and hence requires the use of heuristics. Of course, the addition of the placement problem makes the construction of heuristics even more difficult.

4.2.1 The Assignment Problem
We can define the bistro assignment problem more formally as follows. Given:

- a graph $G = (V, E)$ with a capacity function $c$ defined on the edges,
- a subset $B$ of vertices containing the bistros’s,
- a subset $C$ of vertices containing clients, and
- a path $\gamma(k, j)$ from client $k$ to bistro $j$ for each $k, j$ pair

our goal is to choose one path for each client (i.e., to decide for each client to which bistro it is assigned). Note that, we assume that given a client/bistro pair there is only a single valid path on which a data transfer will take from the client to the bistro. The motivation for this assumption is that our interests are in data transfers over the Internet, and given the current state of the Internet (i.e., IP), in most cases there is a single fixed route between each pair of hosts (i.e., routes do not change other than due to failures).

Once we fix the choice of paths from the clients to the bistro’s we can define several objective functions. Assume that the data rate for client $k$ is $\lambda_k$. Of course, if $\gamma(k, A[k])$ is the chosen path for each $k$ (i.e., client $k$ is assigned to bistro at location $A[k]$) then we have the requirement that for each edge $e$, the data rates on the chosen paths using edge $e$ do not exceed $c(e)$. Then, potential objective functions include:

$$\text{maximize } \left( \min_k \lambda_k \right)$$

(1)

$$\text{maximize } \left( \sum_k \lambda_k \right)$$

(2)

Max-Min Fair Solution

(3)

e.g., given fixed file sizes, Equation (1) corresponds to minimizing the maximum response time (i.e., the response time of the “slowest” client), Equation (2) corresponds to maximizing the throughput, and Equation (3) is intended for addressing of fairness criteria, if such are desired (however, we do not give further details of this objective function here as this is not the focus of this paper).

All three objectives yield interesting classes of problems, unfortunately all three are NP-complete by a reduction from Satisfiability (refer to [1] for details of the proof).

4.2.2 The Placement Problem
The statement of the placement problem is simple — choose the location of the bistro’s to obtain the best solution to the assignment problem stated above. (We can also show that this problem is NP-complete[1].)

4.2.3 Discussion
We note that the issues of service replica placement and selection have been studied in the networking literature, in the context of download applications. Nevertheless, we believe that these problems warrant a study in the context of upload applications as well, at least within the framework presented in this paper. The reason being that the commit and the data transfer sub-problems (as outlined above) are not independent.

5 Conclusions
In this paper we stated (a) a new problem, i.e., the many-to-one communication, or upload, problem as well as (b) the (currently) fundamental obstacles to building scalable wide-area upload applications. We have also proposed a general framework, which we termed the Bistro system, for a class of upload problems, where the basics of the framework are to divide the problem into three subproblems (a) timestamp, (b) commit, and (c) data transfer. Moreover, we have suggested a number of open research problems, within this framework, throughout the paper.

Our long term goal is to accomplish scalability in Internet-based upload applications through the use of the Bistro framework over a wide range of applications and problem sizes. We believe that the Bistro framework is extensible to other Internet-based applications which have a many-to-one data transfer component, such as e-commerce, online auctions, Internet-based storage, and many more. Since the scalability of many-to-one data transfer has not been addressed yet, solving the many-to-one problem will improve the scalability of all these applications.

In general, we believe that there is a need for a scalable infrastructure that will enable Internet-based Computing, where wide-area storage and computational resources are utilized in general large-scale computations. The Bistro architecture is designed to manage storage resources available in the Internet. By viewing wide-area data transfer as a primitive computation, (i.e., a copy), we envision extensions to the Bistro architecture that can also take advantage of computational resources available in the Internet (such as done in [2]) by appropriately augmenting the set of primitive services. By designing, implementing, and deploying Bistro, we will gain knowledge and experience that are fundamental to making Internet-based Computing a reality.

References


