Pixie: A Jukebox Architecture to Support Efficient Peer Content Exchange

Sami Rollins
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
srollins@cs.ucsb.edu

Kevin C. Almeroth
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
almeroth@cs.ucsb.edu

ABSTRACT

Peer-to-peer (P2P) content exchange has recently gained attention from both the research and industrial communities. The dynamic nature of peer networks and the resource constraints of peer hosts have introduced a host of unique technical challenges that must be addressed to make large-scale P2P content exchange applications viable. In this work, we present and evaluate Pixie, an architecture that integrates one-to-many distribution of content and peer networks. Pixie provides a valuable data location service as well as a number of scalability properties both in terms of data location and content distribution. Our results indicate that, using a one-to-many scheme, we can significantly reduce the resources consumed in searching for and distributing content across peer networks. These scalability properties will become increasingly important as peer content exchange is extended to support more advanced applications.

1. INTRODUCTION

Peer-to-peer (P2P) content exchange has recently gained attention from both the research and industrial communities. Systems like Napster and Gnutella launched P2P into the spot light while systems like Chord and CAN have gone a step further in terms of supporting reliable and efficient content exchange. The range of applications that fall into the P2P space has exploded. From distributed computation to distributed file storage, any application that supports cooperation between end hosts is often considered P2P. However, as P2P becomes recognized as more than just the latest buzzword, there is a call to identify and solve the technical challenges that are faced in P2P environments. The dynamic nature of peer networks and the resource constraints of peer hosts have introduced a number of unique technical problems.

In this work, we present and evaluate Pixie, an architecture that integrates one-to-many distribution of content and peer networks. In Pixie, peers join the network and retrieve a *schedule* of content to be distributed. Peers can browse the schedule and choose to take advantage of an already-scheduled distribution. Alternatively, a peer can choose to request that a new distribution be scheduled. When a distribution is scheduled to begin, the serving peer distributes the content throughout the network to all interested peers.

In this paper, we evaluate two properties of Pixie. Pixie differs from traditional P2P in that it provides a new data location service. Our schedule of files currently being delivered, or about to be delivered across the network serves as a browsable index of content. In addition, this schedule can reduce searching overhead by up to 59.1%. Next, we look at the scalability benefit Pixie provides with respect to content distribution. By aggregating client requests and using one-to-many, batched content distribution, we greatly reduce the wait time of clients as well as the use of resources such as disk space, distribution time, and bandwidth on the serving peer.

This paper is organized as follows. In Section 2 we define peer-to-peer and expand on the challenges facing P2P content exchange. In Section 3 we look at current solutions to the challenges facing P2P applications. Section 4 presents our architecture and Sections 5 and 6 evaluate the benefits of the architecture we propose. We conclude in Section 7.

2. MOTIVATION

In this section, we first look at the current impact of P2P content exchange and then address some of the limitations facing P2P networks.

2.1 P2P Content Exchange

P2P encompasses a huge area, from distributed computing [1] to collaborative applications [2]. Applications such as classroom educational tools that enable users to communicate are often considered P2P regardless of their implementation. Alternatively, tools such as application-layer multicast [3] that are implemented using a P2P model, yet support a variety of applications, also fall within the P2P space. In this work, we focus on P2P content exchange applications. This includes the tools, protocols, and applications that support exchange of content between end users.

Napster's pioneering efforts spawned a number of academic and industrial projects aimed at developing efficient, P2P content exchange applications. The primary use of these applications has been the exchange of MP3 music files. But, factors such as increased disk space and higher bandwidth are enabling exchange of other forms of media such

as digital video. As more peers increasingly send more and larger files, a number of challenges become apparent.

2.2 Challenges of P2P Content Exchange

As the range of P2P applications increases, P2P content exchange faces a number of challenges:

- **Peer Discovery and Group Management** - The dynamic, ad hoc nature of peer groups makes it difficult to implement peer discovery and group management algorithms. Centralized solutions largely defeat the purpose of a peer network and can be too restrictive if a centralized infrastructure is not available. On the other hand, distributed solutions generally require a great deal of overhead in terms of state kept about other peers and messaging required to maintain that state.

- **Data Location** - The distributed nature of peer networks makes data location a difficult problem. Having a centralized index or catalogue of available content again defeats the purpose of a P2P solution and may not be possible if no centralized infrastructure exists. At the other extreme, a fully replicated index wastes resources at each peer and would be difficult, if not impossible, to maintain in a dynamic environment.

- **Reliable and Efficient Exchange** - End-user peers are inherently resource constrained. Especially when compared to centrally administered servers, end-user devices (e.g., desktops, laptops, or PDAs) are restricted with respect to bandwidth, disk space, processing power, as well as up-time since peers cannot be relied upon to remain connected for any specific length of time. This limitation makes reliable content exchange more challenging in the P2P environment. New and innovative schemes must be employed to provide fast downloads and avoid overloading the resources of peers that store *hot* items.

3. RELATED WORK

In this section, we discuss current approaches to solving the challenges facing P2P content exchange applications.

3.1 Peer Discovery and Group Management

Discovery and management of peer groups can be implemented using a centralized solution, a distributed solution, or a hybrid solution. Centralized solutions such as those used in Napster, Magi, and Groove are most efficient because peers need not keep state about other peers. Moreover, peers can locate each other with a single request to the centralized directory. The problem with this approach is that it requires a centralized infrastructure. Such an infrastructure may not always be available or may introduce a central point of failure which minimizes the benefit of a P2P system.

Distributed solutions such as Gnutella, FreeNet [4], Chord [5], CAN [6], Tapestry [7], and Pastry [8] generally rely on using a well-known peer to discover the rest of the peer group. However, the group management protocols employed by these solutions are distributed. In Gnutella and FreeNet, a peer keeps track of a constant number of other peers. This is efficient in terms of the state kept at each peer. The problem with the approach is that searching the peer network may be slow.

Chord, CAN, Tapestry, and Pastry represent the second-generation of P2P protocols. In each of these protocols, the network is organized such that peers keep track of a logarithmic number of other peers (with respect to the number of peers in the network). When searching, the protocols can guarantee, or guarantee with high probability, that the desired item can be located in a logarithmic number of peer hops.

There has also been some exploration into the tradeoffs between centralized and distributed solutions [9]. A group of peers, particularly those using mobile devices, may have intermittent access to a centralized infrastructure. Therefore, it may be beneficial to have the ability to tradeoff between centralized and distributed solutions based on the currently available infrastructure.

Peer discovery and group management has so far been a primary focus of P2P content exchange research. We believe that the research in this area is promising. Therefore, in this work, we focus our attention on the following two aspects of content exchange.

3.2 Data Location

Most of the work on supporting data location in peer networks has focused on on-demand searches for information. Systems like Gnutella and Napster, as well as CFS [10], OceanStore [11], PAST [12], systems built on top of Chord, Tapestry, and Pastry respectively, allow the user to search for a particular document. The user must know the name of the document prior to requesting it and searching is then performed on-demand. While many of these systems claim to support file system-like functionality, the infrastructures do not support file system-like content location. Providing that kind of support would require the application to keep track of metadata about each user's files. Even so, this facility would not support exchange of content between users.

Users may not always have a target item they wish to download. Our solution provides users with catalogue of content that they can *browse*. The benefits of organizing content into a browsable catalogue include both a more pleasurable user experience as well as reducing the bandwidth a user consumes when searching for information.

3.3 Reliable and Efficient Content Exchange

In the P2P space, techniques for making content exchange more reliable and efficient have relied on replicating data within the network. Most deployed systems such as Napster and Gnutella rely on the assumption that data are inherently replicated throughout the network. First, the user selects the best peer from which to download content. If the download request fails, generally because the other peer is not reachable, the user must try a different peer.

This model begins to break down when *hot* data is stored on only a small number of peers. Especially if the peers are resource-constrained, they may not be able to support multiple simultaneous requests from the remainder of the network. This problem is further exaggerated by the fact that peer networks are often composed primarily of *freeriders* [13, 14], peers that are only part of the network long enough to retrieve content from other peers.

As peer networks grow, and as multimedia content becomes larger and consumes more resources such as disk space and bandwidth, a more efficient scheme for exchanging content is required. In this work, we focus on the latter two challenges. Pixie addresses the problem of data location by providing a browsable catalogue of popular content available across the network. Moreover, the catalogue caches the location of content making data location more resource efficient. Additionally, we address the challenge of efficient exchange of content by batching requests for content and servicing hundreds or thousands of requests simultaneously. From the client perspective, this greatly reduces the wait time experienced after issuing a request. From the server perspective, we greatly reduce the resources required at the serving peer including disk space, distribution time, and bandwidth.

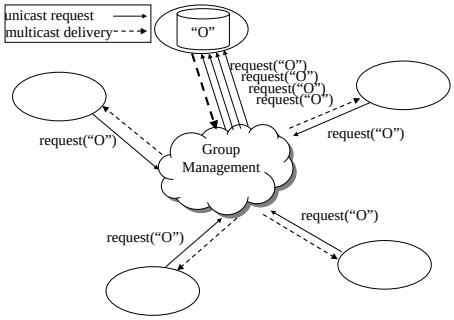

Figure 1: Overlapping requests are aggregated at the serving peer.

4. PIXIE SYSTEM DESIGN

To overcome many of the challenges of traditional P2P content exchange systems, we explore using one-to-many content distribution in peer networks. In this section, we discuss the motivation of our work, provide an overview of Pixie, and discuss the architecture in more detail.

4.1 The AIS

Our architecture is inspired by the Active Information System (AIS) [15], a near-on-demand architecture to support scalable content delivery. The AIS batches client requests for content and produces a schedule of the content to be disseminated. When a client *tunes in* to the system, the client may choose to receive content already scheduled, or may choose to schedule a new distribution. The tradeoff in this case is the time the user must wait to receive content. Dissemination is done using multicast thus relieving much of the burden on the network.

The AIS batching paradigm is well-suited for P2P content exchange. By batching download requests and distributing content to multiple peers in parallel, we can ease much of the burden placed on the peer acting as a server as well as the network. Additionally, the schedule of content to be distributed acts as a *hot list* catalogue. Users can consult the schedule as a means to browse content available in the network.

Unfortunately, the current design of the AIS is targeted toward centralized, video-on-demand style applications [16] and is not well-suited to deployment in a P2P network. All content is distributed by a centralized, fixed set of servers. This is certainly not the case in peer networks. Therefore, we have borrowed the AIS paradigm to create an extended architecture to support efficient, scalable, P2P content exchange.

4.2 Pixie Overview

In this work, we introduce Pixie, an architecture to support one-to-many distribution of content in peer networks. The first goal of Pixie is to aggregate peer requests to download content and use intelligent, one-to-many delivery (e.g., multicast) to enable a large number of peers to take advantage of the same distribution (see Figure 1). The second goal is to publish a schedule of content to be distributed to allow users to *browse* through the most popular subset of content available across the network. Pixie can be implemented on top of virtually any peer group management protocol. When a peer joins the network, it requests the *schedule*. The schedule contains information about content that will be distributed (e.g., Gone with the Wind), how the peer is to receive the content (i.e., the IP address of the multicast group), and when the distribution is scheduled to begin (e.g., 8pm GST). If a user is not interested in content already scheduled for distribution, the user may choose to search for and schedule new content. When a new distribution is scheduled, a *scheduleUpdate* is propagated to all peers indicating the name of the content that will be distributed, how an interested peer can receive the content, and the scheduled distribution time. At distribution time, interested peers *tune in* to the distribution.

Using this model, peers are able to more rapidly and efficiently locate data of interest. The schedule provides a new service, acting as a browsable hot list of available content within the network. Assuming that many users are interested in the same content, it is likely that a user will find the content he or she is interested in by looking at the schedule, thus easing the burden on the network.

By distributing content using one-to-many distribution, we provide additional scalability properties as well. Efficiency gains come from reducing the load on peers by aggregating requests and servicing multiple peers simultaneously. At the scheduled time, the sending peer distributes the information using one-to-many distribution. All interested peers simply *tune in* and receive the content. While network-layer multicast is the most efficient distribution mechanism, we also envision the use of application-layer proxies to reach peers that may not be multicast capable.

4.3 Pixie Architecture

Figure 2 shows the general architecture of a Pixie peer. The Pixie components are implemented on top of a group management layer. We place no restrictions on the group management protocol. We envision anything from Napster-style centralized management to Gnutella-style distributed management to Chord-style distributed management. We discuss each component in more detail:

ScheduleManager. The ScheduleManager controls access to the schedule. The schedule contains information about which data are scheduled to be distributed, when distribution will begin, and where the data will be distributed. It is the equivalent of a TV guide that indicates which programs will be showing, at what time, and on which channel. Each peer retrieves a copy of the schedule when joining the network. Where the copy is found depends on the group management algorithm employed. In a Napster-style net-

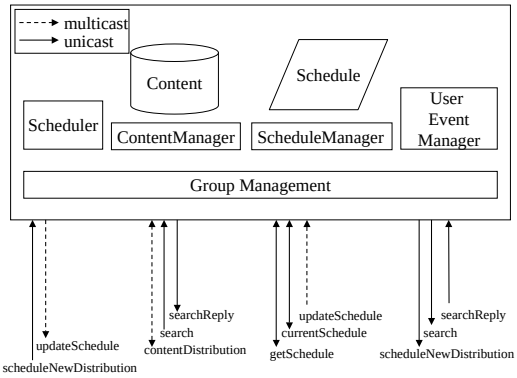

Figure 2: Architecture of a Pixie peer.

work, a getSchedule request will be routed to the centralized server. In a Gnutella-style network, a getSchedule request will be routed to a neighboring peer. We consider the schedule to be *best effort* in that we do not guarantee the peer will receive the latest version. However, if a peer receives a stale version and attempts to search for or schedule an already-scheduled piece of content, the peer serving the content will simply respond with an update indicating where and when the content is already scheduled. The ScheduleManager also receives and applies any updates to the schedule. Schedule updates contain relevant information about newly scheduled distributions (i.e., the content to be distributed, when the distribution will begin, and where the data will be distributed).

Scheduler. The Scheduler handles the scheduling of content distribution for a given peer. When the Scheduler receives a request for new content, it determines when the peer will have the resources available to fulfill the request. For example, if a peer can only support two simultaneous distributions and it is already distributing two streams, the new distribution must wait at least until one of the distributions has finished. The Scheduler may also apply more advanced scheduling algorithms such as delaying distribution in anticipation that more peers will be interested in the same content in the near future. Once the distribution has been scheduled, an updateSchedule message is generated and sent to all peers in the network. The most straightforward method of distributing the updateSchedule message is via multicast. However, a broadcast or gossiping scheme could be used propagate the message.

In a decentralized system, the Scheduler will exist on each peer and each peer will be responsible for scheduling distribution of its own content. However, a centralized implementation could also be employed. In a Napster-style system, a centralized authority would have information about each peer and could make scheduling decisions based upon that global information. This could be more efficient in terms of resource usage, however would require the presence of a centralized infrastructure.

ContentManager. The ContentManager controls access to the data stored on each peer. If a peer is not interested in scheduled content, it can search the network for other content. Search requests are routed through the network in a manner consistent with the underlying group management protocol. For example, using a Napster protocol, search requests would be routed to a centralized server while in a Gnutella protocol, requests would be routed to neighboring peers. When a peer receives a search request, it consults its content base and returns information about content matching the search query.

The ConentManager is also responsible for distributing content. At the scheduled time, the ContentManager distributes the content, preferably using multicast. Application-layer multicast distribution [3] can be employed for peers without multicast connectivity. Additionally, we could employ a digital fountain-style scheme [17] both for efficiency and reliability. In a digital fountain scheme, the ContentManager distributes data encoded using Tornado codes. The serving peer continuously distributes until the client peer has received enough encoded content to reconstruct the file. An additional benefit of using a digital fountain scheme is that clients with new requests can take advantage of distributions already in progress. The penalty incurred is that the serving peer must then continue distribution until the new client has received enough content to reconstruct the file. Finally, the ContentManager is responsible for receiving and storing content distributed by other peers.

It is possible that a peer will leave the network even though it is scheduled to distribute content. On one hand, we argue that Pixie can be used in environments such as within an enterprise where machines are more likely to be well-behaved and remain up. However, in the case that a machine does go down unexpectedly, we rely on the user to reissue a request to schedule content for distribution. Using more robust group management protocols such as Chord, we could imagine automatically rescheduling distributions when peers are discovered to be no longer reachable.

UserEventManager. The UserEventManager processes events from the user and interacts with the user interface. It initiates searches for content specified by the user, requests new content be scheduled, and receives and displays search responses. This component is quite flexible and can be implemented to suit the preferred user interface.

5. EVALUATION OF DATA LOCATION

In this section, we evaluate the benefit of using Pixie for data location. Unlike other systems, Pixie allows users to browse an index of the most popular content available within the peer network. This property provides an enhanced user experience while incurring minimal overhead. Moreover, our approach utilizes fewer network resources by eliminating the requirement that the user must search the entire network for every piece of content.

5.1 Metrics

We are interested in three metrics:

1. **Found** - Found describes the frequency that the user finds an item of interest in the schedule. This metric will allow us to conclude how useful the schedule is for users.

2. **Aggregated** - Aggregated describes the frequency that the user is interested in an item in the schedule *and* can take advantage of the scheduled distribution. This metric provides insight into how often multiple users are serviced by the same stream. We explore resource savings in more detail in Section 6.

3. **Schedule Size** - The size of the schedule will help us to determine the amount of repetition in the schedule and the manageability of the catalogue.

5.2 Setup

To evaluate these metrics, we have simulated the *schedule* portion of our architecture. When a request is made, we schedule the request according to the following algorithm:

```
if the requested item is scheduled
  record as FOUND
  if the distribution has not started
    record as AGGREGATED
  else if the distribution has started
    schedule at end time of current distribution
else
  schedule at current time + 1 minute delay
```

We assume distribution is done through a basic reliable one-to-many distribution service such that users can only join the distribution from the beginning. This service can be either a network-layer multicast group or can be an application-layer distribution service. An item remains in the schedule from the time it is scheduled until it has been distributed.

This model does not entirely capture two cases: (1) the case when multiple peers distribute the same content simultaneously and (2) the case when scheduling incurs an additional delay because a peer's resources are otherwise occupied. Our argument in the first case is that Pixie obviates the need for multiple peers to distribute content simultaneously. No more than two sequential distributions are ever scheduled, thus the maximum wait time is never more than the length of a single distribution. Enabling parallel distributions could be achieved by caching, in the schedule, a list of peers that are capable of distributing a piece of content. Using this technique, we could also avoid the case that a single peer becomes *known* for distributing a particular piece of content.

In the case that a peer incurs additional delay before scheduling an item, we claim that the model we use is, in fact, the most restrictive for the metrics we consider. Lower delay means that items remain in the schedule for a shorter period of time and are less likely to be *found* or *aggregated*. Incurring an additional delay because a peer is distributing other content or is otherwise busy would only improve our results.

We generate a trace of requests using a Zipf distribution [18]. Recent studies have shown this to be typical for current P2P systems[1]. For all experiments we use a catalogue of 400,000 items and run the experiment for a simulated period of 8 hours. To analyze the behavior of the system, we vary two main parameters:

1. **Load** - We look at the system behavior under different load conditions by varying the number of requests per second made across the network from 20-90. Values are taken from recent studies of the gnutella network [14, 19] that indicate that a single peer services or routes roughly 20 requests per second.

2. **Peer Characteristics** - We look at the behavior of the system based on different peer characteristics by varying the distribution times of the objects. This provides insight into the system behavior with large and small files as well as fast and slow connectivity. Table 1 details the values chosen. Distribution time is chosen randomly between the minimum and maximum times.

Min Time (sec)	Max Time (sec)	Description
1	500	Fast Connection High Variance
10	50	Fast Connection Low Variance
3800	4300	Mid Connection
10800	21600	Slow Connection High Variance
15120	16920	Slow Connection Low Variance
120	180	Typical of Current Usage

Table 1: Results of varying min/max distribution time.

5.3 Results

We present the results of three experiments. In the first experiment, we look at the *found* and *aggregated* metrics with respect to varying the load (number of requests per second) across the network. In the second experiment, we vary the peer characteristics in terms of the time to distribute a single item (the effect of either larger files or peers with slower connections) and again look at both the *found* and *aggregated* metrics. Finally, we look at the *schedule size* metric with respect to a spike in load and varied peer characteristics. We follow with a discussion of the impact of these results.

Figures 3 and 4 illustrate how the number of found and aggregated items changes over time for different numbers of requests per second. For this experiment, we fix the minimum and maximum distribution times at 1 and 500 seconds respectively. We observe that the greater the number of requests per second seen by the network, the greater the number of both *found* and *aggregated* items at each 1 minute interval. This is not surprising since a greater number of requests will mean that the schedule of distributions is larger and there is a greater probability of overlap.

We also observe that, in all cases including the case when the load spikes from 30 to 80 requests per second from minute 120 to minute 180, the number of found and aggregated items stabilizes quickly and remains stable throughout the experiment. This property allows us to conclude that under varying load conditions, the system will remain stable.

Another interesting observation is that the *percentage* of requests that are found and/or aggregated remains relatively stable throughout the experiment. The percentage of found items ranges from 54.0% overall in the 20 requests per second case to 65.1% overall in the 90 requests per second case and the percentage of aggregated items ranges from 43.5% overall in the 20 requests per second case to 54.6% overall in the 90 requests per second case. Thus, we can extrapolate

[1] http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

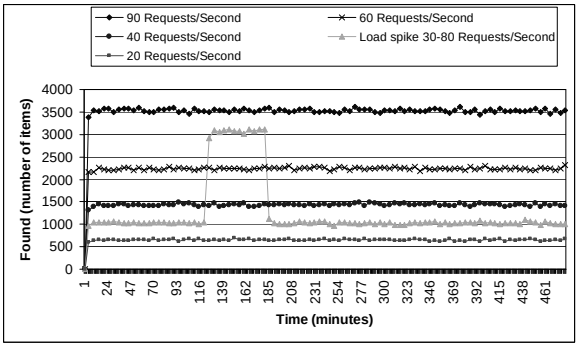

Figure 3: Number of items *found* over time for multiple numbers of requests per second.

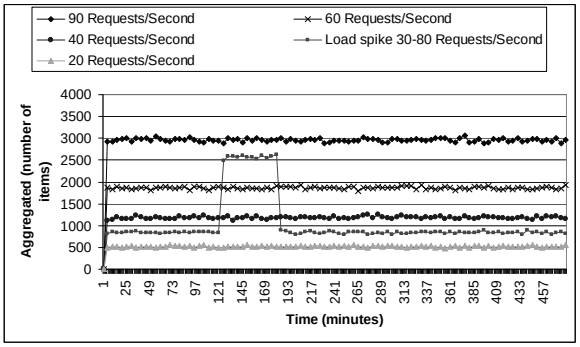

Figure 4: Number of items *aggregated* over time for multiple numbers of requests per second.

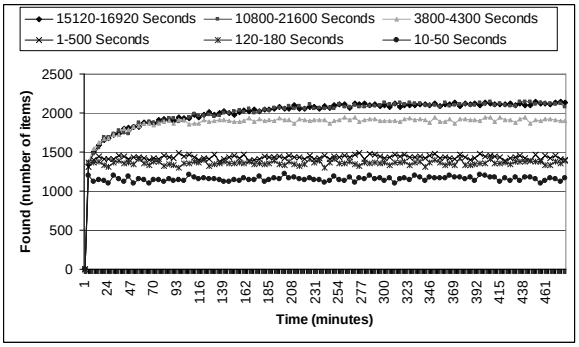

Figure 5: Number of items *found* over time for various distribution times.

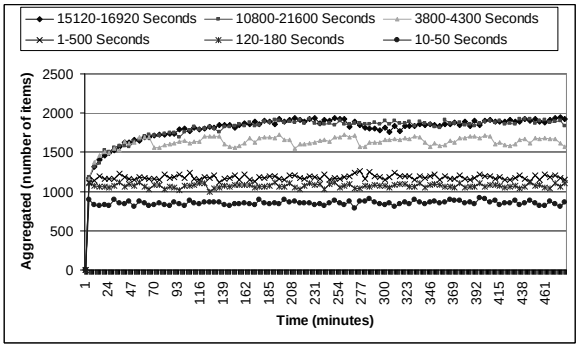

Figure 6: Number of items *aggregated* over time for various distribution times.

that even under varying load conditions, nearly the same percentage of requests will be found or aggregated overall.

Our final observation is that the difference between the number of *found* and *aggregated* items is relatively small. Thus, the case when an already-scheduled item must be scheduled again is relatively rare. Most requests for the same content can take advantage of an existing, scheduled distribution.

Figures 5 and 6 illustrate how the number of *found* and *aggregated* items changes over time for different item distribution times. The item distribution time is the amount of time it takes to distribute a particular item. The greater the distribution time, the greater the number of *found* and *aggregated* items at each one minute time interval. The reason for this behavior is that items with longer distribution times will remain in the schedule longer. Hence, the schedule itself will be larger and the probability of finding an item in the schedule will be higher. Additionally, when items have longer distribution times, the system takes longer to stabilize. This is because no items are removed from the schedule until the initially scheduled items finish.

We also observe that faster distribution times result in fewer found and aggregated items overall. This is simply because when requests are processed faster, there is less opportunity to find a scheduled or executing distribution. Our results indicate that when downloads occur very quickly (10-50 seconds), the percentage of items found in the schedule is 48.1% and the percentage aggregated is 35.6%. This is still a substantial percentage and would still render our system useful.

Our final observation is that slower connections with low variance tend to be quite cyclic. This is largely because the low variance means that all requests initially scheduled are likely to finish at nearly the same time and new requests will be scheduled at that time. This behavior is less likely to occur in a system with varying load, or one in which the load gradually builds up to a stable point.

Figure 7 illustrates the total size of the schedule and number of distinct schedule items over time. We fix the number of requests per second at 40 and introduce a spike in load from 40 to 90 requests per second from minute 120 to minute 180. For the standard 1-500 second distribution time, the schedule size stabilizes quickly, recovers quickly from the load spike, and the size of the schedule is nearly identical to the number of distinct items. With distribution times from 3800-4300 seconds, we see that the schedule takes almost an hour to stabilize initially, does not completely stabilize during the load spike, and is large overall, over 60,000 items at its most stable point. The 6,000-7,000 item schedule for 1-500 second distributions is quite manageable. However, to deal with a larger schedule we might have to employ a solution such as caching only the *hottest* parts of the schedule and asking for other parts of it on demand. We leave this question as future work.

5.4 Discussion

Pixie introduces a new model for P2P content exchange. From a user's perspective, Pixie means that P2P is no longer simply pull-based information exchange. Interesting information is actually *pushed* to the end user. Our results show that, in a moderately loaded system, over 59.1% of search

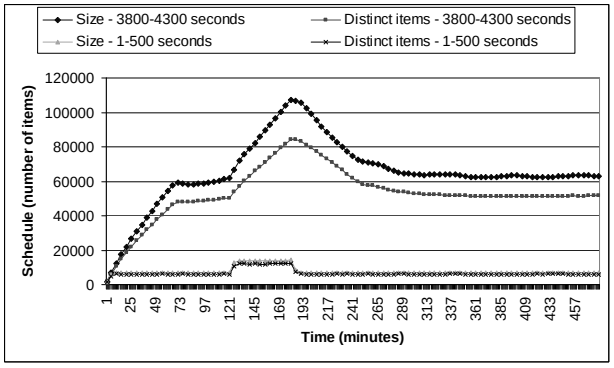

Figure 7: Total *size of schedule* and number of distinct items in schedule.

requests can be satisfied locally. This not only improves the user experience, it greatly reduces the number of search messages flowing in the network. However, our approach does incur some additional penalty. First, since only 48.7% of queries can be aggregated, roughly 51% of queries will result in a scheduleUpdate message that must reach all peers in the network. Fortunately, this penalty is more than offset by the search savings, especially in a decentralized network where multiple peers must not only route search requests, but must process and respond to them as well.

The results we have presented attempt to model today's peer networks as accurately as possible. Our results indicate that the benefit of Pixie is clearly affected by factors such as the load on the peer network, the size of the files being exchanged, and the properties of the connections between the peers. In addition to providing a new data location service, we gain efficiency in terms of the number of messages exchanged throughout the network. By batching distribution of content, we also gain efficiency in terms of the resources on the serving peer. Quantifying serving peer efficiency gains is the subject of the following section.

6. EVALUATION OF DISTRIBUTION

In this section, we quantify the benefit of aggregating requests and batching content distribution in P2P networks and compare multiple schemes for scheduling content. Using Pixie, we can reduce the load on the serving peer as well as the wait time of the requesting peers.

6.1 Metrics

We are interested in two primary metrics:

1. **Wait Time** - In order to evaluate the benefit aggregation provides to the client, or requesting peer, we look at wait time. Wait time describes the amount of time the client must wait from the time it requests a piece of content until the distribution begins.

2. **Number Serviced per Distribution** - To evaluate the benefit aggregation provides to the serving peer, we look at the number of clients satisfied with each distribution. Using this metric, we can extrapolate on the resource savings of using an aggregation scheme.

6.2 Setup

To evaluate these metrics, we have simulated a single peer receiving and servicing requests. Since most peers act in a similar manner, modeling a single peer will provide us with adequate information to evaluate the desired metrics. If the peer receives a request for content that is already scheduled but has not begun, that request is aggregated and will be serviced by the already-scheduled distribution. If the request is for content that is not scheduled or is already being distributed, the peer schedules a new distribution of the requested content. We compare three scheduling schemes with respect to our target metrics.

We generate our traces using the parameters outlined in Section 5. Unless otherwise noted, experiments are run for 480 minutes, item distribution time is between 1 and 500 seconds, 40 requests per second are made across the entire network, and the serving peer stores one piece of moderately popular content. Of the 40 requests per second made across the network, only those requests for the content stored on the serving peer will be processed. All scheduling is done first come first served with respect to the requests for content. We discuss each of the scheduling schemes evaluated in more detail:

- **FCFS** - This is the base case, *first come, first served*, no aggregation-no delay scheme. Distribution is one-to-one as is the case is current P2P systems and requests are serviced as soon as they are received.

- **AGG-<DELAY>** - This is an *aggregation-delay* scheme. Multicast distributions of requested content are scheduled with delay <DELAY>, specified in minutes.

- **DF-<DELAY>-<MAXDIST>** - This is a *digital fountain-delay-maximum distribution time* scheme. Digital fountain style distributions [17], as described in Section 4, are scheduled with delay <DELAY> as in the previous scheme. In addition, since the digital fountain scheme can cause starvation if requests for the same content continue to arrive, <MAXDIST> is a variable that specifies the maximum number of times a single distribution can be extended.

In addition to varying the scheduling scheme, we vary the following characteristics to evaluate system behavior:

1. **Load** - As in Section 5, we look at behavior under varying load. We look at loads of 40 and 90 requests per second across the system. Additionally, we look at the behavior of the system during a spike in load.

2. **Peer Characteristics** - Also, as in the previous section, we look at the system behavior based on peer characteristics with regard to the distribution time for each item. In addition, we vary the type of content stored on the serving peer between popular (many requests made for the content) and unpopular (few requests made for the content).

6.3 Results

We present the results of four experiments. In the first experiment we look at our target metrics in the average case. The second experiment evaluates variations in terms of load and peer characteristics by varying the number of requests

per second made across the network as well as the popularity of the content stored on the serving peer. In the third experiment we take a slightly different look at the wait time metric while varying the load across the network. Finally, in the last experiment we look at both metrics with respect to the peer characteristics by increasing the time required for distributing a single piece of content.

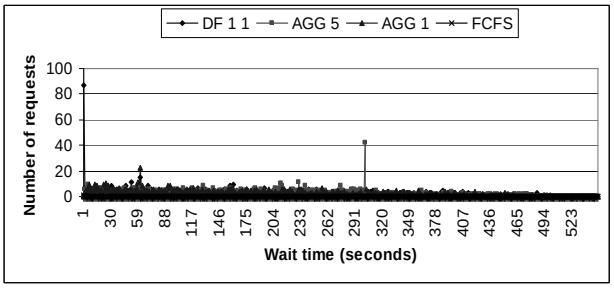

Figure 8: Number of requests experiencing each wait time at 40 requests per second.

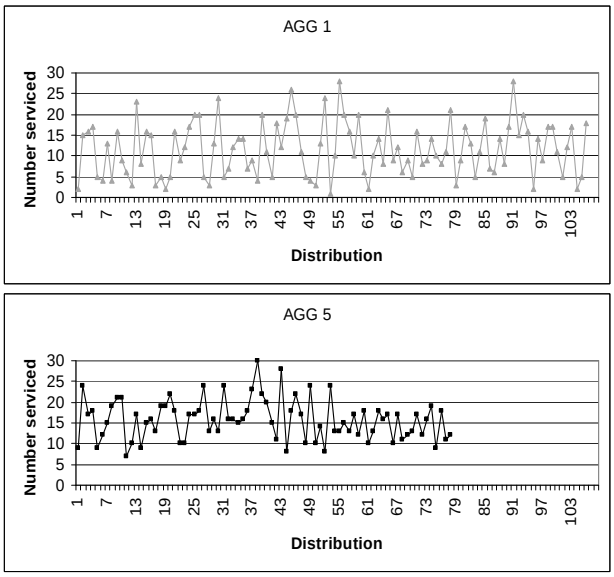

Figure 9: Number of requests serviced with each distribution at 40 requests per second.

Figure 8 plots the number of requests that experience each given wait time throughout the 480 minute experiment. In the FCFS case, many requests are made, queued, but not serviced within the 480 minute window. This is a common occurrence and illustrates the instability of the system using a FCFS scheme. Unserviced requests will remain in the queue of waiting requests at the end of the 480 minutes and we do not report on them here. We have truncated the FCFS data for presentation, but what happens in the FCFS case is that most of the serviced requests are issued in the first few timesteps. Because they are serviced sequentially, each request waits from the beginning of the experiment until the time it is serviced and the time waited increases linearly for each serviced request. In fact, the final request serviced waits for 25,487 seconds.

While the FCFS wait times increase linearly, in all aggregation schemes compared, the wait time remains relatively constant. Using aggregation, no request ever waits for greater than 500 seconds because, in the worst case, a request will be issued just as a distribution is starting, hence the request will have to wait the duration of the distribution. This worst case would be affected if the serving peer stored more than one piece of content. In the worst case, a peer storing N pieces of content would schedule them sequentially, *1, 2,...,N*. If a request for *1* was issued right after the distribution began, the request would have to wait $\sum_{i=1}^{N} distribution\_time_i$ seconds until *1* was scheduled again. A peer could potentially distribute multiple pieces of content simultaneously, but the time to complete each distribution is still restricted by the peer's outgoing bandwidth. Additionally, we suggest that by enabling users to browse a schedule of content, requests are likely to be influenced by content already scheduled.

Another observation of Figure 8 is that the spikes at wait times 0, 61, and 301 indicate that the largest number of requests wait for the amount of time specified by the delay of the aggregation scheme. This is because the system is somewhat lightly loaded and relatively few requests arrive between the time that a distribution is scheduled and when it begins. While a longer delay implies a longer average wait time, the tradeoff is that a longer delay scheme utilizes fewer resources at the serving peer.

Finally, the difference between straightforward aggregation and a digital fountain-style aggregation scheme is minimal. This is primarily because we delay any new distributions by 1 minute and because we restrict the number of times the distribution can be extended to 1. In fact, since our serving peer in this experiment stores only one piece of data, in an unrestricted digital fountain scenario we would achieve 0 wait time for all requests. If a peer was stable, likely to remain available, and stored only 1 or a few pieces of popular content, using an unrestricted digital fountain scheme would be the best choice.

Figure 9 illustrates the number of peers serviced for each distribution scheduled during the 480 minute run using the same parameters used for the experiment shown in Figure 8. We omit the results of the DF 1 1 scheme for presentation since the results were similar to the AGG 1 scheme. We also omit the results of FCFS because it always services one request.

We notice that the aggregation schemes manage to service up to 30 requests per distribution. We also notice that the AGG 5 scheme never services less than 7 requests per stream while the AGG 1 scheme often services fewer. Because AGG 5 consistently services more requests than the lower delay scheme, fewer distributions are required. This is simply because more requests are aggregated prior to the beginning of a given distribution. What this implies is that there is a tradeoff between the resources used at the serving peer and the wait time experienced by the client. By incurring an average wait time penalty of 33 seconds with the AGG 5 scheme, we gain roughly a 25% resource savings at the serving peer. Our final observation is that by using aggregation we gain an advantage in terms of disk space required across the network. For the FCFS scheme to achieve the same performance of the AGG 5 scheme, content would have to be replicated up to 30 times throughout the network.

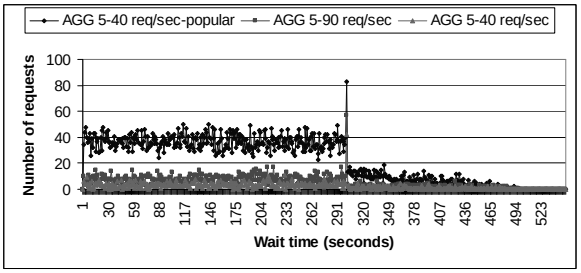

Figure 10: Comparison of the number of requests experiencing each wait time at 40 and 90 requests per second.

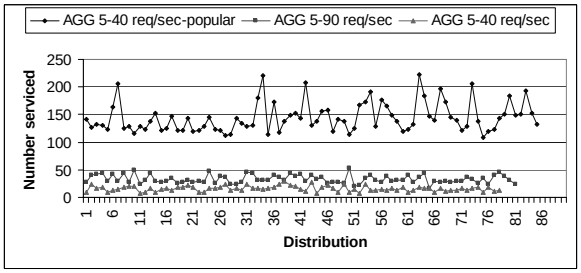

Figure 11: Comparison of the number of requests serviced with each distribution at 40 and 90 requests per second.

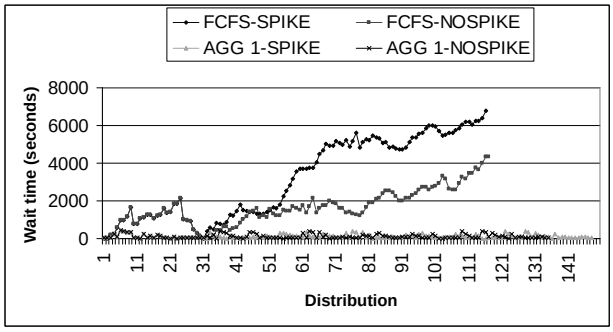

Figure 12: Wait time for each request serviced with load surge.

Figures 10 and 11 illustrate the number of requests experiencing each wait time and the number of requests serviced per distribution respectively for 40 and 90 requests per second and for the 40 requests per second case when more popular content is stored at the serving peer. For the aggregation scheme, increasing the number of requests per second or the popularity of the content has a similar effect. Varying these parameters does not increase the worst case wait time experienced for each request, but rather simply increases the number of requests that experience each given wait time. Based on our evaluation technique, the results of the FCFS case would not change greatly between the 40 and 90 request per second cases, or when the content becomes more popular, because all requests serviced would be issued near the beginning of the experiment. What would change is that the queue of waiting requests would be much larger, though we do not evaluate this metric here.

Additionally, as more requests are issued we can see a clearer pattern. There is an even distribution of wait time from 0 to the amount of delay used in the aggregation scheme. This is largely because requests arrive at a constant rate and are queued until distribution begins. The few requests that experience a wait time greater than the delay are those that arrive while a distribution is in progress and must wait the delay amount plus the remainder of the current distribution.

In Figure 11 we observe that the number of requests serviced per distribution becomes much larger when the load is heavier. Again, increasing the number of requests per second and increasing the popularity of the stored content has a similar effect. Over 12,200 requests are satisfied using and AGG 5 dwarfing the less than 120 requests that can be satisfied in a FCFS case.

In Figure 12 we store an unpopular piece of content on the serving peer and demonstrate a spike in load from 40 to 90 requests per second. The figure illustrates the wait time experienced under these conditions. While wait time with the FCFS scheme continues to grow, even without the load spike, wait time using the aggregation scheme remains stable over all requests. Such behavior is especially important when a new, popular piece of content is introduced into the peer network. In the best case FCFS scheme, distributing a single piece of content throughout the entire network would be logarithmic with respect to the number of peers. Using aggregation, the same distribution occurs in constant time.

Figures 13 and 14 illustrate the effect of extending the amount of time a single distribution takes to complete. This would be the result of distributing larger files and/or using a slower connection. What we notice is that, overall, wait time for the aggregation schemes increases to be, at most, nearly the length of the distribution. This is simply because requests made when a distribution is already in progress must wait until the distribution finishes. With FCFS, the behavior is nearly the same as in previous cases with the maximum wait being nearly the length of the experiment (480 minutes). Looking at Figure 14, we notice that in the FCFS case, only a total of 7 requests are serviced while over 150 requests are serviced with each distribution in the other schemes. The disk space savings Pixie achieves becomes clearer in this case. Assuming files of 5MBytes, a conservative estimate for for content such as video, we can achieve a savings of nearly 1GByte. In a wireless and/or small device environment where bandwidth and disk space are scarce and distribution can be lengthy, an aggregation scheme provides a clear benefit over a straightforward FCFS technique.

7. CONCLUDING REMARKS

In this work, we introduce an architecture to improve data location and content distribution in peer-to-peer networks. In Pixie, peer requests for content are aggregated. A schedule of distribution times is propagated throughout the network, and distribution is done using a one-to-many, multicast scheme. Thus, a virtually limitless number of peers may take advantage of the same distribution.

This scheme provides two primary benefits over straightforward, one-to-one content exchange. First, the schedule of distributions acts as a browsable index of content available across the peer network. Unlike current solutions, users need not know know the name of the content they wish to

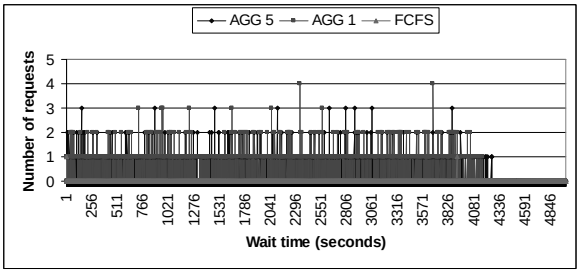

Figure 13: Number of requests experiencing each wait time for content with 3800-4300 second distribution time.

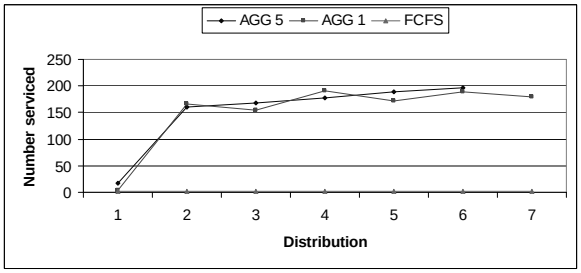

Figure 14: Number of requests serviced with each distribution for content with 3800-4300 second distribution time.

download when searching. Further, the push-based schedule distribution can eliminate search overhead by up to 59.1%. The second benefit of our scheme is more efficient content distribution. By aggregating requests and batching distributions, we significantly reduce the wait time a peer experiences and the resources required on the serving peer(s). Resources include disk space, distribution time, and bandwidth.

Peer content exchange is becoming more popular and encompasses an increasing number of applications. The content being exchanged is becoming larger while the peer devices are becoming smaller. One-to-one distribution is these scenarios is inefficient if not impossible. Moreover, locating content in larger and more diverse networks is even more of a challenge. Using Pixie, we can reduce the impact of these challenges and make content exchange more efficient.

8. REFERENCES

[1] M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability issues in global computing," *Concurrency: Practice and Experience*, vol. 12, pp. 727–753, 2000.

[2] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," Tech. Rep. HPL-2002-57, Hewlett Packard Laboratories, 2002.

[3] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV*, (Port Jefferson, NY, USA), June 2001.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, (Berkeley, CA, USA), July 2000.

[5] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrisnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Sigcomm 2001*, (San Diego, CA, USA), Aug. 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Sigcomm 2001*, (San Diego, CA, USA), Aug. 2001.

[7] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley.

[8] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Middeware*, (Heidelberg, Germany), Nov. 2001.

[9] K. Nagaraja, D. Milojicic, and S. Rollins, "Adaptive infrastructure for mobile ad-hoc communities," in *submitted to ICPP 02*, 2002.

[10] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *SOSP 2001*, (Banff, Canada), Oct. 2001.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *ASPLOS*, (Cambridge, MA, USA), Nov. 2000.

[12] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent, peer-to-peer storage utility," in *SOSP 2001*, (Canada), Nov. 2001.

[13] E. Adar and B. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, Oct. 2000.

[14] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *MMCN*, (San Jose, CA, USA), Jan. 2002.

[15] S. Rollins, R. Chalmers, J. Blanquer, and K. Almeroth, "The active information system (ais): A model for developing scalable web services," in *Internet Multimedia Systems and Applications*, (Kauai, Hawaii, USA), Aug. 2002.

[16] K. Almeroth and M. Ammar, "The interactive multimedia jukebox (imj): A new paradigm for the on-demand delivery of audio/video," in *WWW7*, (Brisbane, Australia), Apr. 1998.

[17] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Sigcomm*, (Vancouver, British Columbia), pp. 56–67, Sept. 1998.

[18] G. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley, 1949.

[19] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal, Special Issue on Peer-to-Peer Networking*, vol. 6, no. 1, 2002.