The Livny and Plank-Beck Problems: Studies in Data Movement on the Computational Grid *†‡

Matthew S. Allen
Computer Science Department
University of California, Santa Barbara

Rich Wolski
Computer Science Department
University of California, Santa Barbara

Over the last few years the Grid Computing research community has become interested in developing data intensive applications for the Grid. These applications face significant challenges because their widely distributed nature makes it difficult to access data with reasonable speed. In order to address this problem, we feel that the Grid community needs to develop and explore data movement challenges that represent problems encountered in these applications. In this paper, we will identify two such problems that we have dubbed the Livny Problem and the Plank-Beck Problem. We will also present data movement scheduling techniques that we have developed to address these problems.

1 Introduction

In recent years, many applications have developed that demand access to large amounts of data. Scientific applications like particle physics [20, 1], fluid modeling [9], and others, require access to terabytes of information. Often, these applications work with data sets that are either too large to be stored at a single site, or are distributed among a group of cooperating organizations, or both.

For these reasons, efficient movement of large data sets is the subject of much current research in Computational Grid computing [6, 14, 11, 5, 13, 18, 12, 22, 16]. The chief performance goals of these systems are to provide the fastest possible remote access and ensure that the access is reliable.

Research on scheduling this data management has focused on both the problem of distributing the storage load among a set of servers and on replication as a way of ensuring reliability and data proximity. In order to store large data sets and keep their load balanced across many hosts, many applications choose to divide these sets into sections and distribute them. To access these files reliably in spite of individual host failures, these sections are frequently replicated across many file servers.

While the projects cited above have each explored these problems in different ways, commonalities among the various successful solutions are beginning to emerge. In this paper, we investigate two such commonalities, identified by noted researchers in the field: Dr. Miron Livny [4] from the University of Wisconsin, and Dr. James Plank [2] and Dr. Micah Beck [3] from the University of Tennessee, Knoxville. During various conversations and collaborative activities with each of them, they have independently posed separate challenge problems that we believe are germane to the field at large. As such, we have named each problem after its progenitor. Briefly described, the *Livny Problem* focuses on optimizing a set of independent transfers to a given target location with the goal of improving the time to arrival of the greatest number of transfers. The *Plank-Beck Problem* involves the dynamic construction of a data stream from replicated and distributed stream segments. We describe these problems in greater detail and investigate how effective dynamic scheduling can be used to address them.

Thus, the contribution that this paper makes is twofold:

- Begin the process of identifying relevant community challenge problems for data intensive Grid applications.
- Investigate efficient scheduling techniques in terms of their ability to address these challenges.

The first contribution, we believe, is a necessary step toward the definition of realistic and pertinent benchmarks for Grid computing. To date, despite various efforts to de-

† SC'03, November 15-21, 2003, Phoenix, Arizona, USA Copyright 2003 ACM 1-58113-695-1/03/0011...$5.00

‡ This work was supported, in part, by NSF Awards, CAREER 0093166, EIA-9975020, and EIA-9975015, the DOE SciDAC program.

velop viable benchmarks [23], a well-accepted set remains elusive. We believe that challenge problems, independently formulated and addressed, will aid these efforts, particularly for data intensive applications. With network capacity as the bottleneck resource, efficient use is imperative to promote both individual application performance and the scalability of the Grid to multiple competing applications. Our scheduling work, based on the network monitoring and forecasting information provided by the Network Weather Service [26], attempts to detail effective ways of addressing these problems using currently available and supported Grid infrastructure.

In particular, this paper will describe a methodology we developed for selecting replicas and scheduling file transfers using NWS measurements and forecasts. We will compare algorithms that use these measurements with other solutions to these scheduling problems. We demonstrate our results in terms of two applications we have written: an unordered file aggregation application modeled on the Livny Problem and a data streaming application with replicated segments modeled on the Plank-Beck Problem. We will formally describe the Livny and Plank-Beck Problems in section 2, and we will describe our algorithms for addressing them in section 3. Section 4 details our experiment's implementation and section 5 will show our results. We conclude with section 6.

2 The Livny and Plank-Beck Problems

The Livny and Plank-Beck Problems are generalizations of common requirements that arise frequently in data intensive Grid applications. They are both motivated by specific needs within the Condor [21, 15] project lead by Dr. Miron Livny at the University of Wisconsin and the LBONE [12] project headed by Dr. Micah Beck and Dr. James Plank at the University of Tennessee, Knoxville. In this section, we will formally define these problems and we will describe the applications that best embody them.

The Livny Problem stems from a discussion of network monitoring and forecast efficacy that took place between one of the authors and Dr. Livny during a Grid research workshop. Livny posed, as an investigative challenge, the problem of optimizing unordered collective downloads to a single location from a set of remote locations. More precisely,

> *Given k files of uniform size that are distributed across k hosts in a Grid setting, optimize the transfer of all files to a central location so that the maximum number of files arrive in the minimum amount of time.*

The goal is to schedule the downloading of these files such that the largest number of files are downloaded in the shortest period of time. In this situation, it is impossible to improve the time it takes to download the entire collection of files since "slow" ones must necessarily be slow (i.e. there is no replication). Notice also that stated in this form, "tree" reductions in which a file may make multiple hops (as a log function of the number of hosts) are also possible. After discussing the problem further with Dr. Livny, we determined that for this initial investigation, all communication channels had to be directly between the individual remote sites and the collection site.

Although no specific application was presented by Dr. Livny in this challenge, we speculate that it is related to checkpoint file management. Condor supports automatic process checkpoint and migration when resource owners reclaim their resources. Since hosts are being reclaimed, it is not clear that they would be available to participate in a large tree reduction of checkpoint files, or that the parallelism such a reduction would bring would be beneficial (e.g. if the hosts were connected to a shared network). As such, we chose the simpler formulation of the problem in which each remote host is constrained to communicate directly with the collection site.

The Plank-Beck Problem was taken directly from a demonstration made by Dr. Plank during SC02 in both the National Partnership for Advanced Computational Infrastructure (NPACI) and University of Tennessee (UTK) research exhibits. The application, developed by Plank and Beck's group at UTK, uses an XML registry to keep track of segmented MP3 or DVD files in which the segments are replicated across Internet Backplane (IBP) depots [17]. An IBP-enabled player automatically fetches the segments in the order required to play the stream back sequentially [10, 19]. Moreover, the player begins playing as soon as the first segment arrives and will "stall" when a missing segment is encountered until it is delivered. Thus, the goal of the data movement system is to deliver segments reliably (no segments can be missing) at a rate fast enough to keep the player from stalling.

More precisely

> *Given a single large file divided into k ordered segments of uniform length in which each segment has r replicas distributed across hosts, minimize the time necessary to deliver each segment in order.*

Strictly speaking, this formulation is not representative of the problems faced by Plank and Beck's application. For their application, each segment has a unique deadline that is determined by the segment's relative location in the file and the speed of the player. Thus, minimizing the time for

each segment is not necessary as long as each time proceeds its corresponding deadline. We ignore this requirement to simplify and generalize the challenge problem for this initial investigation. Although our graphs will show some common audio requirements, we will not spend time discussing delivery deadlines in this paper.

We believe the Livny Problem to be representative of a variety of Grid applications that benefit from collecting a portion of a distributed data set as quickly as possible. Also, the Plank-Beck Problem is germane to similar applications in which streaming or ordered delivery of aggregated data objects is essential. At the same time, we do not claim that the Livny and the Plank-Beck Problems completely cover the space of representative challenge problems. Rather, they are the first we have encountered from deployed systems that, in our judgment, have potentially wide applicability. Our goal with this exposition is to encourage the community at large to identify and formulate others.

3 Methodology

For our experiments, we present a set of strategies to address the Livny and Plank-Beck Problems respectively. For the Livny Problem, our scheduling algorithm tries to execute the faster transfers sooner as a way of improving the arrival distribution. For the Plank-Beck Problem, our goal is to provide fast transfer performance without the need for wasted transfers.

To expose the efficacy of our solutions to the Livny and Plank-Beck Problems, we try to improve on some strategies suggested as "control" examples. For the Livny Problem, we followed the suggestion of Livny himself, who speculates that our solution will not be able to substantially outperform a *random* scheduling algorithm. Given the dynamics of network transfers, it is possible that simply choosing hosts at random is a performance optimizing strategy. We test our strategy against this possibility at Dr. Livny's request.

For the Plank-Beck Problem, we compare our results with two algorithms implemented for the application originally deployed at SC02. The first is a *greedy* approach which issues parallel fetches for each segment as a way of optimizing throughput and maximizing fault tolerance. This method retrieves each segment quickly because when it downloads all segments at once the fastest arrives first. It also exhibits good fault tolerance because even if one download fails, another fetch will still deliver the data.

The second algorithm implemented as part of the original Plank application is an improvement on the first that addresses the problem that arises when multiple downloads overload the network, dubbed the *progress-driven redundancy* algorithm [19]. This algorithm does not use parallel streams to fetch multiple replicas of the same segment in parallel. Instead, it fetches different segments from multiple replica servers. The number of threads fetching segments is equal to the degree of replication (ie. with 5 replica sites there are 5 threads fetching segments). If one of the fetches lags and enough segments past it have been downloaded, more fetches are issued for the lagging segment. In this way, each segment is downloaded only once unless it begins to hold up the stream.

However, for correctness, this version of the Plank-Beck algorithm requires that segments within each replica be co-located. That is, files must be stored in their entirety at each replicating site. The algorithm uses a work-queue model in which fetching threads take responsibility for an unfetched segment on the queue, fetch that segment from their replica, dequeue the segment identifier when the fetch is complete, and acquire another segment to fetch. "Fast" replicas will deliver more segments than "slow" ones since their threads will "grab" more of the unfetched segments.

By requiring all segments of a given replica to be stored on the same server, this scheme limits the size of the file that can be replicated to the minimum available space at any of the potential replication sites. For example, if there are four servers available to replicate a file, and they have 10, 20, 30, and 40 megabytes of available space respectively, only a 10 megabyte file could be replicated across all four servers. This storage architecture severely limits the flexibility and scalability of the system.

Thus, we also investigate a *general progress-driven redundancy algorithm*. In this version, each segment is replicated across sites chosen from the host pool for the experiment. Like the algorithm described above, the scheduler maintains a pool of threads and a work queue of segments that have yet to be downloaded. Also, as before, threads "grab" work from the queue until there are no more segments to transfer. In this generalized case, however, when a thread decides to download a segment, it chooses a replica server storing that segment at random since it does not have access to proximity information. If we need to duplicate a fetch, it chooses another server at random and begins a fetch from there. We implement both the original algorithm and this general algorithm in this paper.

We believe that with a different approach we can optimize performance and eliminate the need for parallel fetches in these Plank-Beck solutions. Our approach is based on network performance measurements and forecasts generated by the Network Weather Service (NWS) — a distributed Grid middleware service designed to support high-performance dynamic application scheduling. NWS sensor processes on each machine measure the availability of resources like bandwidth, cpu, and memory using periodic,

non-intrusive tests. Other NWS applications are used to analyze a measurement series and generate a forecast using a set of non-parametric statistic techniques. These forecasts can be used to predict how available a resource will be in the future. Our methods are based on NWS predictions of available bandwidth.

For the purpose of selecting a file for download, this means that we can use the NWS bandwidth predictions to determine which files are "closest" to the application that needs them. A file is said to be closer than another if the link to that file provides higher bandwidth than the other. In many of these scenarios, access times can be improved by giving priority to the files that will arrive the quickest. The proximity of a file can be discovered in many ways, but our success in the past addressing similar problems with NWS predictions suggests that they are effective.

For the Livny Problem, we provide two algorithms we wish to explore. These algorithms compare our prediction based approach with an approach that uses explicit application-level information on download times. Our control algorithm generates a random ordering of hosts to download from. This algorithm would be the performance of the file aggregation with no scheduling policy.

Our first algorithm, called the *last-time* algorithm, attempts to use information gathered at the application level to schedule file downloads. This algorithm records the download times from each host during a run of the application. It then schedules the file downloads based on the last file download time for each host, from fastest to slowest. There are inherent advantages and disadvantages to using application level information. Using performance data gathered at the application level means that the information will be fine tuned to the program's requirements. However, a scheduler based on instrumentation data alone might not have a recent antecedent to base its schedule on.

Thus, the second Livny Problem algorithm, referred to as the *NWS-prediction* algorithm, uses the NWS to determine which host will be the closest to the collection point. It schedules the file downloads based on the NWS prediction of bandwidth to the file servers, from highest to lowest. Like using application level data, there are pros and cons to this approach. Using external data from the NWS requires additional setup overhead and the measurements may not be indicative of true application usage. However, it provides consistent and complete information independent of the application's recent behavior. Predictions are also less susceptible to anomalies like routing changes or network partitions that may affect one download, but not others.

For Plank and Beck's greedy algorithm, we use predictions to eliminate the need fetch the same section in parallel. In contrast to Plank and Beck's greedy algorithm, ours downloads only one file at a time based on which server the NWS predicts has the highest bandwidth connection. This is able to find the closest replica without causing contention between multiple downloads. To achieve fault tolerance, we select the next closest replica in the case of a failure. The tradeoff is that the time spent detecting a failure and fetching another segment is lost.

However, as mentioned previously, part of Plank and Beck's goal is to ensure that each segment arrives and does not stall. In our work, we use dynamic timeout discovery [8] as a way of ameliorating transient network failures. The original Plank and Beck solution works in the presence of machine failure, but failure of a host is rare compared to the "failure" of a particular socket connection. By dynamically learning how long to wait for a connection to complete, our system can automatically retry slow connections. We investigate how effective this strategy is at ensuring application throughput.

To compare with Plank and Beck's progress-driven redundancy algorithm, our second *NWS-prediction* algorithm is modified slightly. For this version, we create a number of threads equal to the number of replicas just as in their algorithm. Each thread then chooses the first segment that no other thread is downloading and fetches the predicted fastest replica. No parallel fetches are issued because it assumes it is retrieving the closest replica.

4 Experiment

To respond to the challenges presented by the Livny and Plank-Beck Problems and to test the usefulness of NWS predictions, we implement two experiments. Both these test applications are based directly on these two problems, and are written to make use of different scheduling strategies. The first is a file aggregation application based on the questions posed in the Livny Problem. The Plank-Beck Problem is addressed by an application that downloads a file stream from a set of replicated stream segments.

Both experimental applications follow the same format. At the core is a scheduler application that runs on the collection point and aggregates files. This scheduler implements algorithms to order downloads to optimize for the Livny or Plank-Beck Problems. This scheduler then contacts file servers located on all the machines in the host pool to download files. The file servers are very simple processes that are distributed across all hosts in the experiments. When a client establishes a connection with the file server and requests a file, it replies with the file's contents. The scheduler takes measurements at every step of its run time so we can analyze the various algorithms' effectiveness. There are 50 file servers for the Livny experiments, and 48 for the Plank-Beck experiments.

4.1 Network Weather Service

For all of our experiments we set up NWS network sensors on each of our file server machines. Each sensor in the experiment pool runs periodic tests with a sensor that runs on the collection point. These tests consist of 64KB network probes run every 30 seconds. Also on the scheduler's host is an NWS forecasting demon, which uses measurements made by the sensors to calculate performance predictions for each of the file server hosts. The scheduler queries this forecaster for NWS bandwidth predictions for the file servers.

We have deliberately chosen to use short NWS network probes in this experiment as a way of determining their resolution power. Probes of this size, particularly for network connections with large bandwidth-delay products, do not give accurate absolute estimates of available network throughput, but the amount of additional network load they introduce is insignificant. They are also not representative of the size of the files we are downloading. However, the forecast required is a forecast of relative rank rather than absolute throughput speed. In these experiments, we wished to investigate the ability of short probes to make rank forecasts. The goal is to keep NWS network probes as small as possible to support scheduling and resource allocation, even if this makes them non-representative of true application usage.

4.2 The Livny Problem

To investigate the Livny Problem, we use a file aggregation program that is implemented as follows. There are 50 hosts in the experiment that run file servers, and each stores a file 10 megabytes in size. The centralized scheduler downloads these files according to one of the scheduling policies. The scheduler accumulates all of these files and aggregates their contents. The experiment completes when all 50 files arrive at the the scheduler.

For these experiments our goal is to test the various scheduling strategies for downloading the fastest files first. The scheduler generates a list of files sorted on what it believes will be the fastest to the slowest. The last-time scheduler uses application level data gathered at the immediately preceding random run of the experiment. This means that all last time measurements are as fresh as possible. Although it is unreasonable to assume that you will have immediately fresh last-time measurements, we implement this primarily to understand the effectiveness of the NWS-prediction strategy. The NWS-prediction scheduler sorts all the files based on the freshest NWS predictions as well, and during run time it recomputes the schedule for the remaining hosts periodically. Thus, it also attempts to use the freshest forecasts.

By comparing the NWS-prediction schedule with the last-time application performance schedule, we can test the loss of accuracy caused by the use of a non-intrusive NWS network probe to "read" the network conditions. We assume that the NWS is part of the Grid middleware fabric and, as such, is constantly monitoring network conditions. Thus, an application run at any time will have access to "fresh" NWS data.

As an intermediate step between the Livny Problem and the Plank-Beck Problem, we want to understand how these Livny Problem algorithms work with simultaneous file transfers. The scheduler generates an ordered list of files to be downloaded based on the scheduling strategy. It then spawns some number of threads to begin downloading these files in parallel. Each thread takes the first file off the list that has not been downloaded and is not being downloaded and issues a fetch for it. When the download completes, it fetches the next file on the list that meets the same requirements. Thus, at all times the scheduler is downloading multiple files at once. We ran these experiments with 2, 4, and 8 simultaneous downloads.

4.3 The Plank-Beck Problem

To explore the Plank-Beck Problem, we implement an experiment to test the effectiveness of using predictions for replica selection. In this experiment, our goal is to download a 700Mb file that is distributed across the host pool. This file has been divided into 70 10Mb segments, and each segment is stored on r different hosts. At the beginning of each iteration of the experiment, these replicas are randomly distributed across the replica servers. Each segment is assigned to r randomly selected different hosts. The randomization process attempts to distribute the load evenly across all the hosts, so that if there are 4 replicas and 35 hosts, each host will be responsible for 8 different segment replicas. Our goal is to download these segments in order in the least time possible.

The greedy algorithm is implemented as described previously. The scheduler simultaneously downloads the next segment by beginning a download from all r of the replica sites. When the first download completes, all other downloads are halted and the scheduler proceeds to download the next segment. The NWS-prediction scheduling algorithm only downloads one segment at a time, chosen from the r replica sites based on data from the NWS. When it needs to download a segment, it sorts the r replicas sites from fastest to slowest. It then retrieves the segment from the nearest file server. We run these experiments with r values of 2, 4, 6, and 8.

We also establish a methodology for dealing with failures in this application. When using the greedy algorithm,

errors are handled by default. If one of the downloads fails, the other downloads continue until one of them completes. In the NWS-prediction method, if there is a failure during connection or download, the scheduler downloads from the next fastest replica and downloads the complete segment regardless of how much was downloaded before the failure. The number of failures in individual replica downloads and overall section downloads is recorded.

We implement both the Plank's original and the general progress-driven redundancy algorithm as discussed earlier. Both algorithms behave as follows. The scheduler spawns r threads to fetch segments, where r is the number of replicas of each segment. Each thread selects a segment in the stream to download based on the following requirements. If there is a segment in the stream that has not been completely downloaded and there are three segments that follow it that have, this segment is considered to be holding up the stream. To rectify this, the thread issues another fetch for a different replica of that segment. Only two downloads are allowed per segment, so once two downloads have been initiated for a segment, no more will be issued for it. If there is no segment that is holding up the stream, the thread instead issues a fetch for the first segment that has not been downloaded and is not currently being downloaded.

The general version of the algorithm differs in how the file is distributed and how a thread chooses which host to download from. In the original progress-driven redundancy, r random sites are chosen at the beginning of the experiment, and each holds a complete copy of the file. Each thread is bound to one site, and when it chooses to download a segment it always downloads from that site. This is the version presented by Plank and Beck, and it utilizes the topology because it is likely that a thread bound to a fast server will finish quickly and will eventually start duplicating downloads from a slower server. In the general progress-driven redundancy version, the file's segments are distributed randomly across all hosts as described in the beginning of this section. When a thread decides to download a segment, it chooses a replica site at random and fetches from it. This utilizes topology because when a duplicate fetch is issued for a slow segment, it is likely that it will be from a faster server. Although the general method may not be as strong as the original algorithm, it does represent what we believe to be a more general formulation of the problem that is likely to occur in heavily used Grid environments.

5 Results

The experiments ran as described in the previous section from March to May of 2003. Experiments were run throughout the day so that they would not be limited to a single time period.

The collection point was located at a workstation at UCSB. The file servers were distributed across a set of machines in the U.S and Netherlands. Many of these machines were provided by the Grid Application Development Software (GrADS) Project [7], which hosts a testbed for developing Grid applications. Our accounts were all non-superuser accounts provided graciously by various collaborators. *Table 1* shows our host pool, including the number of hosts we had access to. Also included are statistics drawn from our experiment data on the average time to download a 10 Mb file and the variance in those download times.

Graphically, we present our results throughout the paper using Cumulative Distribution Functions (CDF) of the aggregate data. For all of these graphs, the x-axis represents the time (in seconds) since the beginning of the run of an experiment. The y-axis is the percentage of files that have arrived at the collection point as aggregated across all experiments. Thus, we can say that x seconds into the aggregation, on average y percent of the files have arrived. Using *figure 1* as an example, we see that 200 seconds into the experiment, 36% of the files have arrived using the random algorithm, and 63% have arrived using the NWS-predicted algorithm. Also, we can tell that on average 50% of the files have arrived in 278 seconds using the random algorithm, which takes 103 seconds with the NWS-predicted algorithm.

5.1 Livny Problem

Our initial results focus on the Livny problem to show the effectiveness of scheduling file transfers. We used data accumulated from 326 runs of each algorithm using all different parameters. In this case, we ran each of the 3 algorithms with 4 different thread counts, for a total of 3,912 total aggregations. *Figure 1* shows the results of the Livny problem experiments with only one download at a time. The random ordering produces results much like we would expect, with the percentage of files aggregated linear to time on average. From this graph it appears that there are only two data sets being plotted, which is because both our last time and NWS-prediction ordering produce almost identical results. This means that NWS predictions rank host download times just as effectively as the freshest possible last time values. This result is important to notice because it means that with no knowledge of the application's needs, the NWS provides information that is just as useful as data gathered directly from the application. Furthermore, either algorithm yields a significant improvement and gets 70% of the files in two thirds of the time of the random strategy.

Table 2 extends these results to cover the behavior of parallel sockets, where we actively download more than

Table 1: Experiment Host Pool

Domain	Hosts	Average Download Time	Download Time Variance
University of California, Santa Barbara	14	0.95s	0.02
California State, Northridge	1	7.81s	42.95
University of California, San Diego	12	3.32s	3.70
University of Illinois, Urbana-Champaign	14	16.98s	6.15
Indiana University	1	23.82s	23.42
University of Tennessee, Knoxville	13	14.30s	8.15
University of Minnesota	1	22.49s	444.97
Vrije University in Amsterdam	2	40.50s	55.15

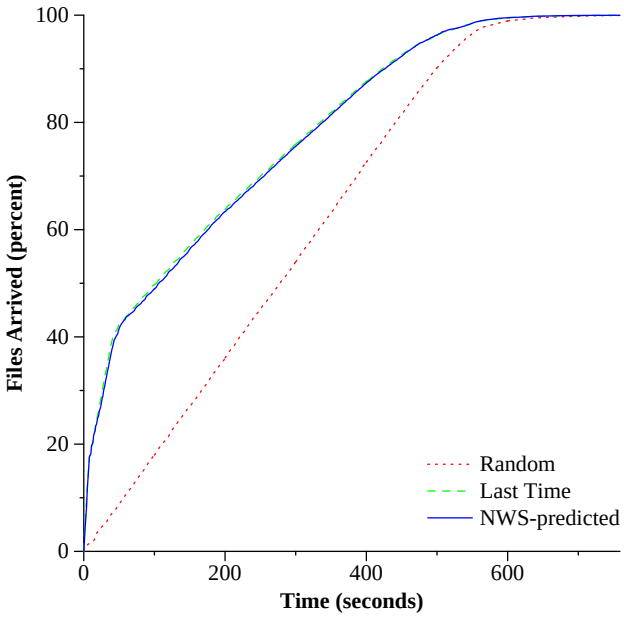

Figure 1: Livny Aggregation CDF (no parallel downloads)

Table 2: Livny Aggregation Times with Simultaneous Downloads (seconds)

	Random	Last Time	Predicted
2 downloads			
25%	69	13	13
50%	132	46	46
75%	196	125	127
90%	231	195	194
4 downloads			
25%	34	13	13
50%	71	34	34
75%	105	80	81
90%	127	115	116
8 downloads			
25%	26	15	15
50%	49	36	36
75%	69	63	63
90%	83	81	81

one file at a time (these results are presented graphically in the Appendix). Each column of the graph shows the time it takes (in seconds) to download a percentage of the files for a specific strategy. For example, the second row of this table shows us that if we keep two simultaneous downloads throughout the aggregation, we will download 50% of the files in 132 seconds with the random strategy or 46 seconds with either the last time or NWS-prediction strategy.

We show these results to demonstrate that the trends we observe in *figure 1* still hold in the parallel transfer situation. As we can see from the graphical representation of this data for 8 downloads in *figure 2*, both strategies still offer an improvement over the random strategy.

It is worth noting that performance for aggregating the fastest files degrades as there is more contention between downloads. This can be seen by comparing the time to reach the 50% quantile n the last time strategy for 4 and 8 downloads in *table 2*. Although it takes 34 seconds to reach this quantile with 4 active downloads, it takes two seconds

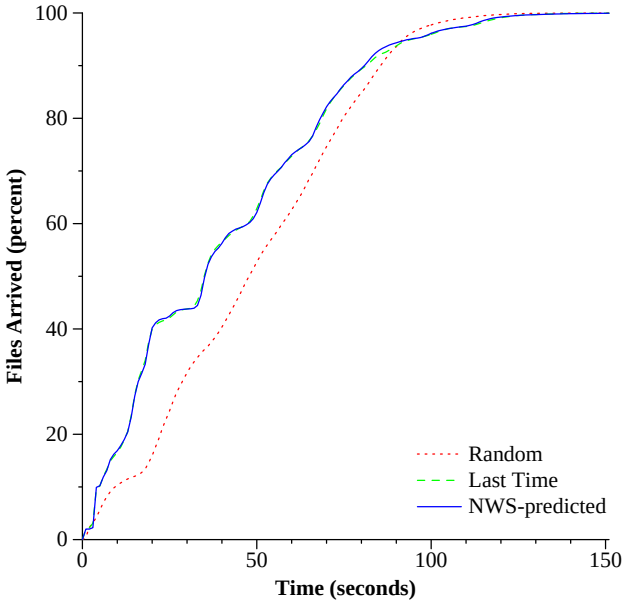

Figure 2: Livny Aggregation CDF (8 parallel downloads)

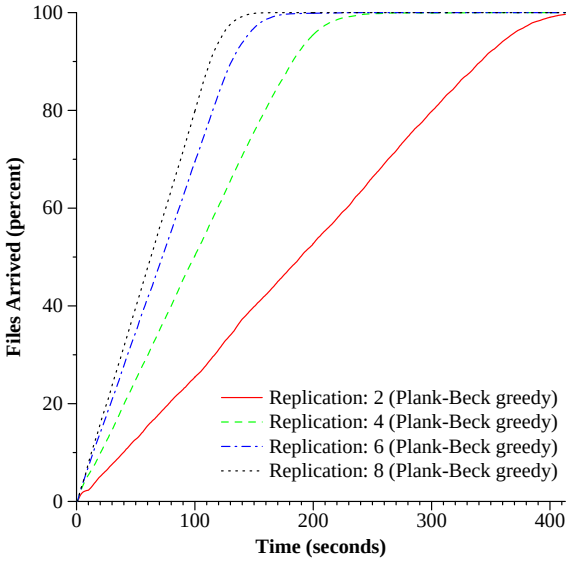

Figure 3: Plank-Beck Aggregation CDF (Greedy only)

longer with twice as many simultaneous downloads. So with 4 downloads, it takes 5.4 seconds to download one of the closest 25 files on average (5.4 seconds * 25 files / 4 download streams = 34 seconds). Doubling the number of downloads means it takes 11.5 seconds on average to download one of the fastest files (11.5 seconds * 25 files / 8 downloads streams = 36 seconds). At this point, doubling the contention for bandwidth at the collection point increases the download time for each of the closest files by over a factor of two. This observation will be applicable to the Plank-Beck problems when we discuss the tradeoff between finding a close file and using multiple download streams.

Based on these results, we believe that NWS proximity information, even with unrepresentative network probes, is a powerful tool for building schedulers to optimize data transfer. For these experiments, scheduling algorithms are able to make a difference in the time it takes to accumulate a large percentage of the total files. This is accomplished with a slight amount of additional bandwidth consumed by the NWS network probes. NWS forecasts do not show better results than a fairly simple application level method. However, the application level method we describe has explicit download data on all the hosts it interacts with as well as exceptionally fresh last time measurements. In a grid setting, hosts would be entering and leaving this operation, and we cannot expect to have information on their previous download times. Also, as the measurements taken during runs become more stale, it is likely they lose their effectiveness as a scheduling metric [25, 24]. Thus, application level data of this quality is not realistic for a Grid setting. For this reason, we consider the fact that NWS-predictions perform as well as this best case application level scheduling a success.

5.2 Plank-Beck Problem

Having established that predictions are an effective metric for scheduling file downloads, we apply these results to the Plank-Beck Problem. The following results are collected from 276 runs each of the Plank-Beck greedy and NWS-predicted download strategies for all parameters. This means with 276 runs of two algorithms with 4 different levels of replication there were 2,208 aggregations in total. For each level of replication, the replicas were randomly distributed throughout the host pool and then both algorithms were run with that configuration. For every experiment using the Plank-Beck greedy strategy, there is an equivalent experiment using the NWS-Predicted strategy.

Figure 3 demonstrates only the Plank-Beck greedy algorithm with various degrees of replication. As we can see, as the number of replicas increases the average download time decreases. The reason is that for each segment it is more likely that there will be a replica close to the aggregation point. *Figure 4* is the same graph except it shows the CDF's for the NWS-prediction based strategy. This figure demonstrates that our method exhibits similar performance benefits for higher degrees of replication.

Table 3 provides insights into how these algorithms accomplish better download times with more replication. The table shows how frequently hosts from each domain were

Table 3: Replica Sites Responsible for Segment Delivery

		Plank-Beck greedy				NWS-Predicted			
domain	hosts	2	4	6	8	2	4	6	8
ucsb.edu	9	34. 17	52. 12	61. 02	67. 32	34. 34	54. 79	67. 07	75. 55
ucsd.edu	11	32. 20	37. 09	36. 17	31. 89	32. 03	34. 41	30. 12	23. 66
uiuc.edu	12	8. 31	0. 50	0. 03	0. 01	8. 39	0. 56	0. 04	0. 01
umn.edu	1	0. 84	0. 18	0. 03	0. 01	0. 79	0. 12	0. 02	0. 01
utk.edu	12	24. 33	10. 12	2. 75	0. 78	24. 32	10. 11	2. 76	0. 78
vu.nl	2	0. 15	0. 00	0. 00	0. 00	0. 13	0. 00	0. 00	0. 00

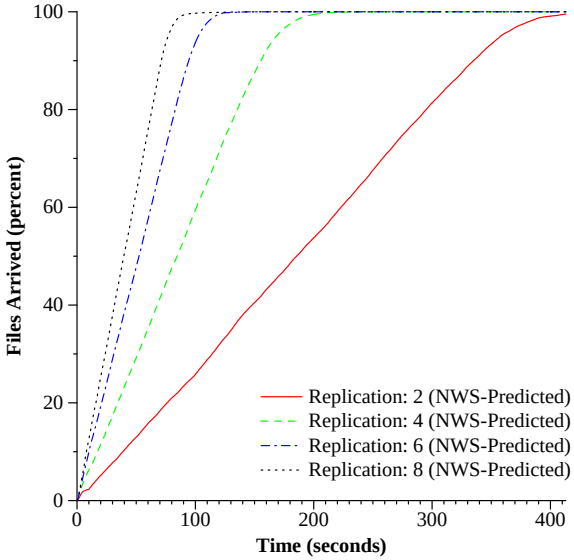

Figure 4: Plank-Beck Aggregation CDF (NWS-Predicted only)

responsible for delivering segments. Here, we can see the number of hosts housing replicas that were used for this experiment on the left. On the right are the percentage of times hosts from those domains were responsible for delivering the segment to the scheduler. Both of the algorithms presented attempt to download the closest replica to reduce the overall aggregation time. For instance, we know from *table 1* that ucsb.edu can deliver a segment 2.4 seconds quicker than ucsd.edu on average. Although ucsd.edu has more hosts and thus is responsible for more replicas (recall that each host is responsible for an equivalent number of replicas), more of the segments were actually delivered by ucsb.edu for all levels of replication. Also, as the level of replication increases, it becomes more likely that there will be a replica on domains close to the collection point (ucsb.edu and ucsd.edu, in this case). Because most of the files are downloaded from these close hosts, the overall run time of the application decreases.

The weakness of the greedy Plank-Beck algorithm is

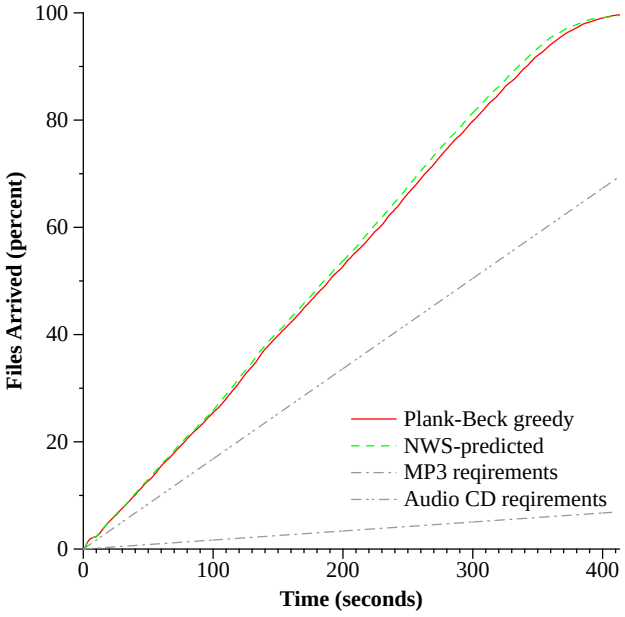

Figure 5: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (2 replicas)

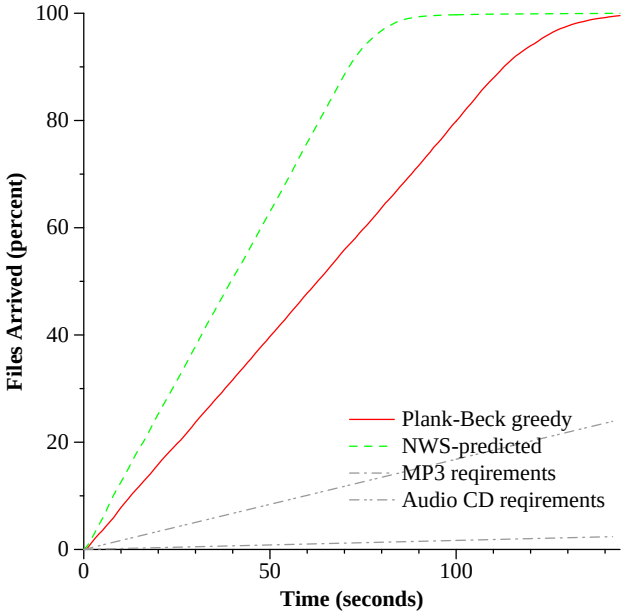

Figure 6: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (8 replicas)

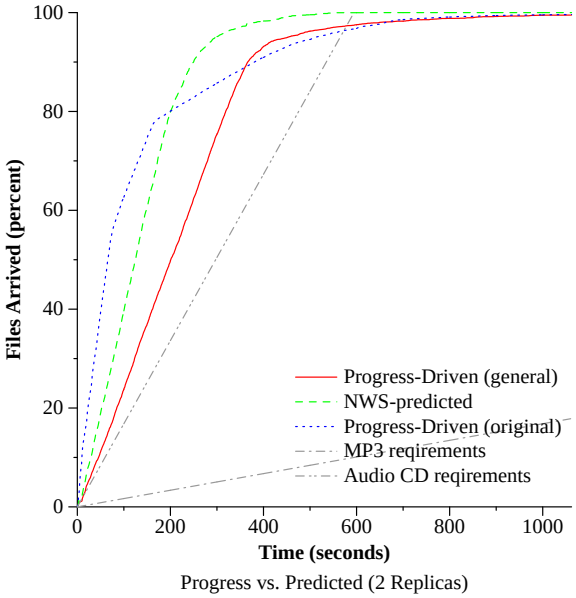

Figure 7: Progress-Driven and NWS-Predicted Aggregation CDF (2 replicas)

Table 4: Plank-Beck Greedy and NWS-Predicted Aggregation Times (seconds)

	Replicas		
	4	6	8
Plank-Beck greedy			
50%	100	73	63
75%	149	108	94
95%	199	144	123
NWS-predicted			
50%	85	53	40
75%	127	78	60
95%	169	103	77

that when we download multiple files at once, there is contention between the downloads. It consumes unnecessary bandwidth downloading data that is discarded once the first arrives. Intuitively, if we download from the closest replica only, our aggregation would gain the speedup from replication without the overhead of multiple streams. When there are a small number of replicas, as shown in *figure 5*, there is a slight but insignificant decrease in aggregation time for the NWS-predicted algorithm because it uses only one download stream. However, with such a low number of replicas we have poor fault tolerance because only two transfers need to fail to make a segment unaccessible.

As we increase the number of replicas, we increase the fault-tolerance and the chance we will have a close proximity file. However, with the greedy strategy increasing the number of replicas increases the contention between downloads and the amount of bandwidth consumed. Looking at *table 4*, we can see that with more replicas the NWS-prediction based algorithm avoids this contention and gains a substantial speedup by only downloading the predicted fastest replica (this information is presented in graphical form in the Appendix). This is most obvious comparing the Plank-Beck greedy algorithm and the NWS-prediction based algorithms for 8 replicas, as shown in *figure 6*. Here we see that the NWS-prediction algorithm reaches each quantile in roughly 65% of the time of the Plank-Beck greedy algorithm. This is because we are able to use the fastest replica while only consuming 12.5% of the bandwidth of the greedy strategy.

In comparison with the Plank-Beck greedy algorithm, our NWS-prediction based algorithm performs quite well. Because the greedy method causes contention between the multiple download streams, choosing only the fastest replica server increases the download speed. Downloading from only one site also consumes a fraction of the bandwidth used by the greedy algorithm. In the worst case, if almost every replicated copy is downloaded in its entirety, then that means the greedy algorithm moves r times as much data across the network (where r is the level of replication). Although this is a worst-case estimate, it still remains the case that the greedy algorithm gains its performance speedup only by inducing more load on the network.

Moving on, we now look at 196 runs of data for the newer progress-driven redundancy algorithm developed by

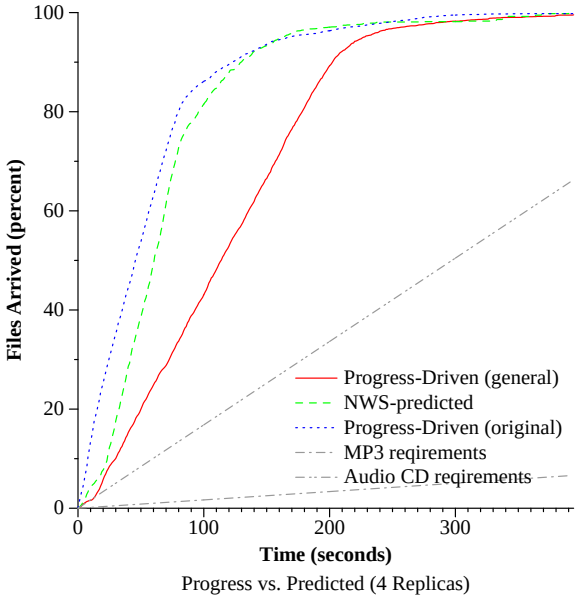

Figure 8: Progress-Driven and NWS-Predicted Aggregation CDF (4 replicas)

Plank and Beck. *Figure 7* shows the performance of our NWS-predicted algorithm against both of the progress-driven implementations for two replicas. In this case, we can see that the NWS-prediction algorithm clearly performs better than the general progress-driven algorithm. This holds true for all levels of replication, and likely means that in the general case (where entire files are not replicated as contiguous segments) the progress-driven algorithm is perhaps not an effective approach. Looking at this graph, it would seem that the original progress-driven performs better than either algorithm in most cases at the expense of the limited architecture discussed previously.

From *figure 8* we can see that the progress-driven algorithm consistently performs better than our NWS-prediction strategy. However, as the degree of replication increases, the performance of the NWS-prediction strategy becomes much closer to the original progress-driven strategy. This is especially true once the aggregation reaches 70%, which is we can see from comparing the 75% rows from *table 5* above (for a graphical representation, refer to the appendix). This is significant because the architecture of the NWS-prediction strategy distributes replicas more widely and evenly. Thus, the NWS algorithm achieves performance similar to the original progress-driven algorithm, but permits segments to be replicated individually (as opposed to contiguously) allowing greater flexibility and scalability.

Based on comparisons with the Plank-Beck greedy strategy and the newer progress-driven algorithms, NWS-predictions appear to provide a strong basis for replica selection. Because predictions are capable of determining proximity, there is no need to consume additional bandwidth with redundant downloads. Although it is hard to draw a strong correlation between the progress-driven algorithms and our strategies, we are able to achieve similar performance for a distributed and scalable storage architecture compared to a more simple and limited architecture. We are still able to accomplish this without duplicating any downloads, unlike any of the strategies that do not use prediction.

Table 5: Progress-Driven and NWS-Predicted Aggregation Times (seconds)

	Replicas			
	3	4	5	6
General Progress-Driven				
50%	140	114	102	95
75%	205	167	147	136
95%	278	227	192	177
NWS-Predicted				
50%	75	61	60	59
75%	111	84	77	72
95%	199	163	137	113
Progress-Driven				
50%	51	46	42	45
75%	91	74	69	73
95%	259	168	131	107

5.3 Failure Characteristics

In our experiments with the Plank-Beck problem our system not result in a significant number of failures, so it is difficult to compare the two algorithms based on fault-tolerance. However, it is worth noting that our experiments did run on a grid testbed designed to characterize a real grid with a production compute load. Thus, it may be the case that programs running in this environment do not suffer enough faults to justify the overhead of parallel downloads. If this is the case, then the best policy may be to support less thorough fault recovery and remain well behaved with regard to other users.

Regardless, if we want to design algorithms that can function in a distributed and faulty environment, we must consider the possibility of failures. Therefore, we run experiments where we introduce transient network failures artificially while we search for “real” network links that exhibit greater loss. To induce failures, we randomly stall 5% of the transfers by a factor of three. For example, there is a 1 in 20 chance that a given download, completing in 10 seconds, will be stalled an additional 20 seconds for a total

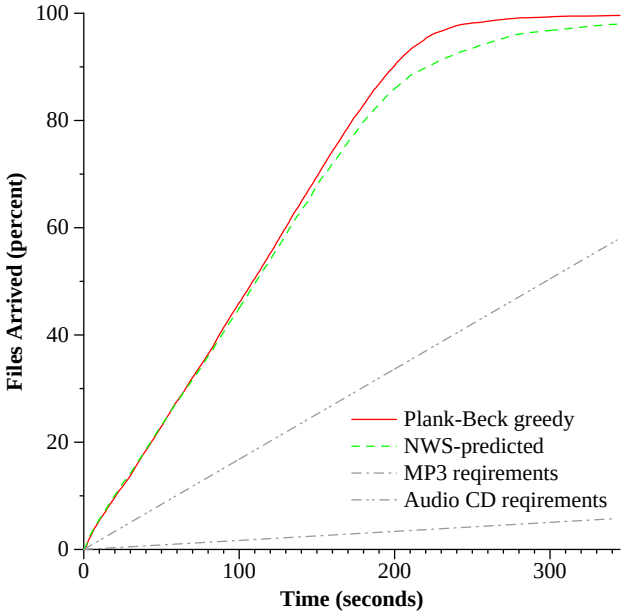

Figure 9: Plank-Beck Aggregation with Induced Errors CDF (4 replicas)

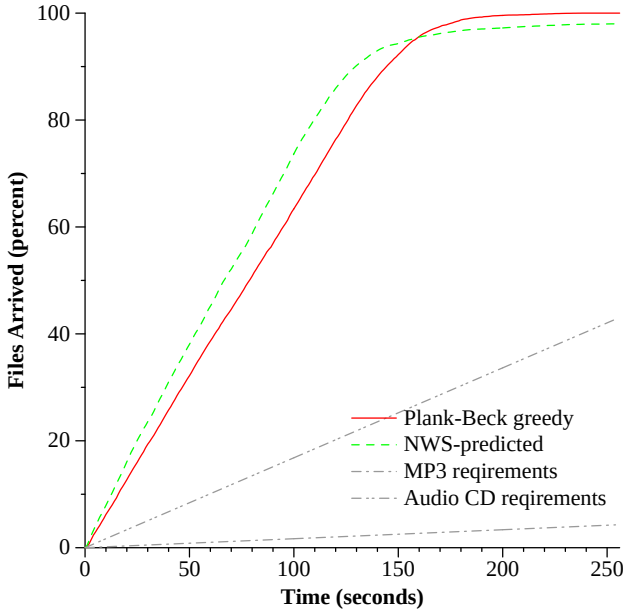

Figure 10: Plank-Beck Aggregation with Induced Errors CDF (6 replicas)

transfer time of 30 seconds. Although this is an artificial and probably overly-aggressive failure model, it attempts to address the types of problems that are reported by Plank and Beck that we did not observe.

Figure 9 shows the effects of these artificially induced errors on the Plank-Beck greedy and NWS-predicted comparison. Here, we can see that the Plank-Beck greedy method performs slightly better in all cases. The NWS-predicted method shows some performance degradation once the aggregation is 95% complete. This is caused by aggregations where some number of downloads have been stalled and the NWS-predictive method cannot respond as quickly as the greedy method. However, this decrease in performance only affects a small portion of the downloads. The performance difference between the two strategies is very slight, and the NWS-prediction algorithm accomplishes this performance using only 25% of the bandwidth of the greedy method and does not violate TCP congestion control mechanisms.

Figure 10 shows the same comparison with 6 replicas. Because of the throughput penalty, in this example the NWS outperforms the Plank algorithm in terms of speed while achieving the same level of fault tolerance. Again, through the use of forecasting and effective retry, the NWS algorithm achieves the same degree of fault-tolerance, uses less bandwidth, and runs faster than the greedy strategy in the average case.

In comparison with the Plank-Beck greedy algorithm, our NWS-prediction based algorithm cannot respond as quickly to failures. The greedy strategy initiates one download per replica, which means that if any transfer fails or is stalled, there is a redundant download in progress that can be used for recovery. However, as the number of replicas increase, the proximity determining benefit from this strategy becomes obviated by the overhead of downloading multiple files. Also, in our experiments with four replicas, the performance of our NWS-prediction based strategy is not significantly worse. Given this, using predictions performs reasonably well and behaves much better in terms of wasting resources than the greedy strategy.

To consider the effects of these error experiments in terms of the progress-driven algorithm, compare *figures 11* and *12* with their counterparts *figures 7* and *8*. What we see from this graph is that these errors do not substantially change the performance of the aggregation except to slow down all strategies roughly equally. Since the progress-driven method attempts to only duplicate downloads that are held back, it is far less aggressive about duplicating streams. Waiting until the a section has fallen behind does not seem to be a more aggressive failure detection method than using NWS-predictions on time to arrival for the same purpose.

Because failure detection is no more aggressive in the progress-driven method, it does not seem to have much advantage over the NWS-prediction method. In fact, all versions of these algorithms suffered roughly the same performance penalties from this type of error. Because of this and because our NWS-prediction method supports more scal-

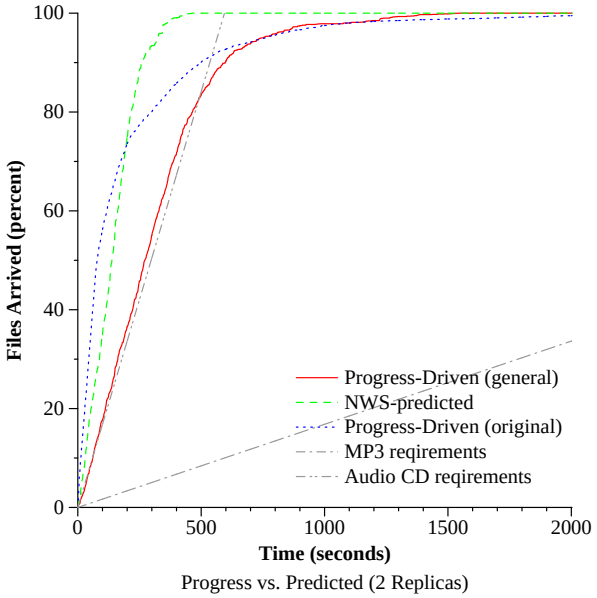

Figure 11: Progress-Driven Aggregation with Induced Errors CDF (2 replicas)

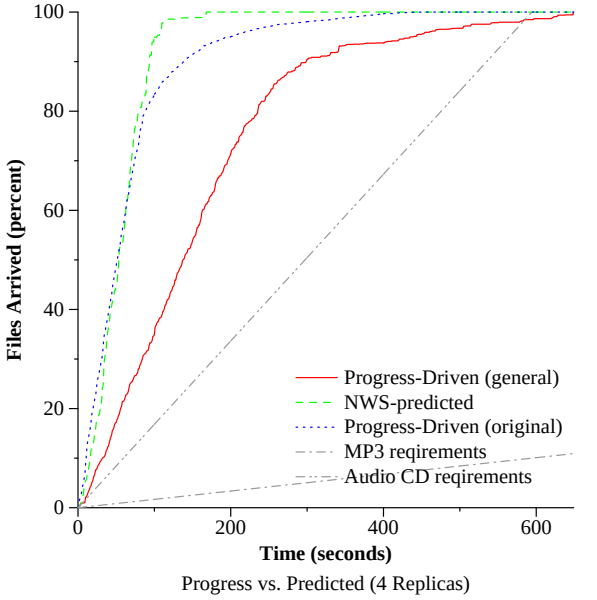

Figure 12: Progress-Driven Aggregation with Induced Errors CDF (4 replicas)

able and distributed architecture, we believe our method is as fault-tolerant and more flexible than the progress-driven method.

6 Conclusion and Future Work

There are many difficulties with efficiently accessing data that is distributed in a wide area network. We have formally described some of these difficulties with two challenge problems–the *Livny Problem* and the *Plank-Beck Problem*. As representatives of current data movement challenges, we feel these problems must be explored to develop efficient data intensive Grid applications.

We present solutions to optimize the goals of these two challenges. When the goal is to access the closest files first, one option for determining file proximity is to use measurements and predictions provided by applications like the NWS. Although it is effective to use information gathered at the application level, using NWS predictions is equally powerful. Also, monitoring systems are capable of providing more complete network information and simplify the program's complexity by handling monitoring responsibilities.

In the related problem of replica selection, predictions can be a useful metric as well. Predictions have the advantage of being able to determine the fastest replica to download. Other methods are either not as effective at ranking hosts or cause contention between multiple downloads. Although predictions may take longer to respond to failures than other methods, the performance benefit from using them makes up for this discrepancy.

Moving forward, there are many other solutions to these problems that need to be understood to develop the Data Grids that are one of this community's current focus points. Although we present solutions that we believe to be quite powerful, this is not the only purpose of this paper. We also want to challenge our fellow researchers to study and explore these problems. By addressing these challenges, we will all help to solve many problems faced by current research projects.

References

[1] CERN homepage. `http://public.web.cern.ch/public/`.

[2] Dr. James Plank's homepage. `http://www.cs.utk.edu/~plank/`.

[3] Dr. Micah Beck's homepage. `http://www.cs.utk.edu/~mbeck/`.

[4] Dr. Miron Livny's homepage. `http://www.cs.wisc.edu/~miron/`.

[5] European Data Grid Project homepage. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[6] Globus Data Grid homepage. http://www.globus.org/datagrid/.

[7] GrADS Project homepage. http://hipersoft.cs.rice.edu/grads/.

[8] M. Allen, R. Wolski, and J. Plank. Adaptive timeout discovery using the network weather service. In *High Performance Distributed Computing*, 2002.

[9] T. Arbogast and Z. Chen. On the implementation of mixed methods as nonconforming methods for second-order elliptic problems. *Mathematics of Computation*, 64(211):943–972, 1995.

[10] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore. Fault-tolerance in the network storage stack. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Ft. Lauderdale, FL, April 2002.

[11] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker, 1998.

[12] A. Bassi, M. Beck, T. Moore, and J. Plank. The logistical backbone: Scalable infrastructure for global data grids. In *Asian Computing Science Conference*, 2002.

[13] M. Beynon, R. Ferreira, T. M. Kurc, A. Sussman, and J. H. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *IEEE Symposium on Mass Storage Systems*, pages 119–134, 2000.

[14] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, 1999.

[15] Condor home page – http://www.cs.wisc.edu/condor/.

[16] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[17] J. Plank, M. Beck, and W. Elwasif. IBP: The internet backplane protocol. Technical Report UT-CS-99-426, University of Tennessee, 1999.

[18] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The internet backplane protocol: Storage in the network, 1999.

[19] J. S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for high performance, wide-area, distributed file downloads. Technical Report UT-CS-02-485, Department of Computer Science, University of Tennessee, October 2002.

[20] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim. Storage managementfor high energy physics applications. In *Computing in High Energy Physics*, 1998.

[21] T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.

[22] D. Thain, J. Basney, S.-C. Son, and M. Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, August 2001.

[23] R. F. V. D. Wijngaart and M. Frumkin. Nas grid benchmarks version 1.0. Technical Report NAS-02-005, NASA Advanced Supercomputing, July 2002.

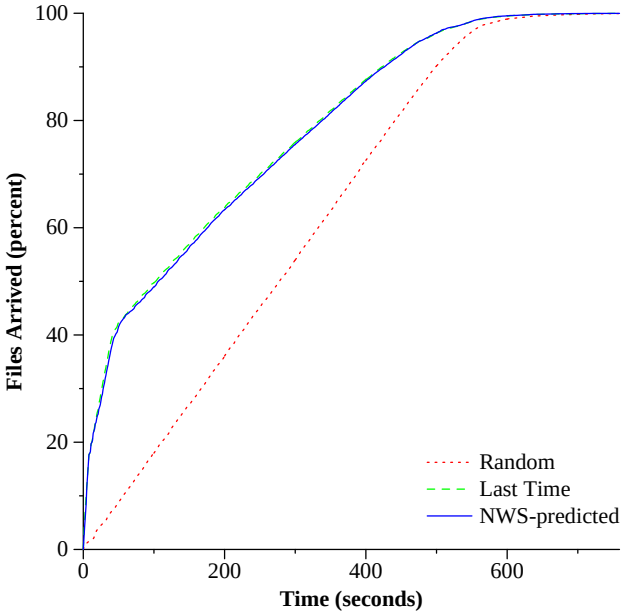

Figure 13: Livny Aggregation CDF (no parallel downloads)

[24] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, 1998. also available from http://www.cs.ucsb.edu/~rich/publications/nws-tr.ps.gz.

[25] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance EvaluationReview*, 30(4), March 2003.

[26] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999. http://www.cs.ucsb.edu/~rich/publications/nws-arch.ps.gz.

Appendix

Figures 13 and *16* are duplicates of *figures 1* and *2*. *Figures 14*, *15*, and *16* graphically represent the data displayed in *table 2* in the results section. These figures show the random, last time, and NWS-predicted scheduling strategies for the Livny problem. Each CDF displays the percentage of files that have arrived at any point in the aggregation's run time for different numbers of parallel downloads. These graphs show results for 1, 2, 4, and 8 simultaneous downloads.

Figures 17 and *20* are duplicates of *figures 5* and *6*. *Figures 18*, *19*, and *20* graphically represent the data contained in *table 4* in the results section. These figures show the

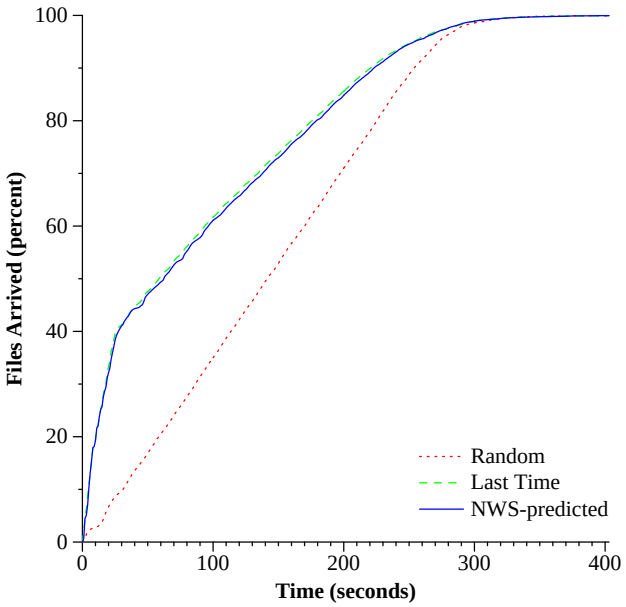

Figure 14: Livny Aggregation CDF (2 parallel downloads)

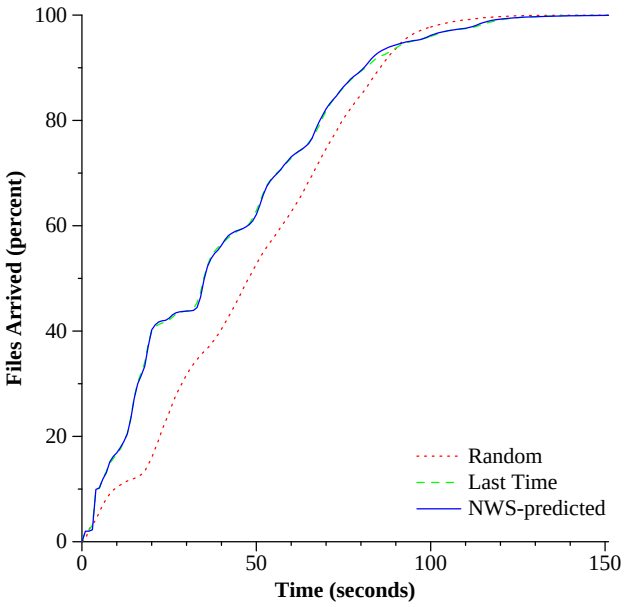

Figure 16: Livny Aggregation CDF (8 parallel downloads)

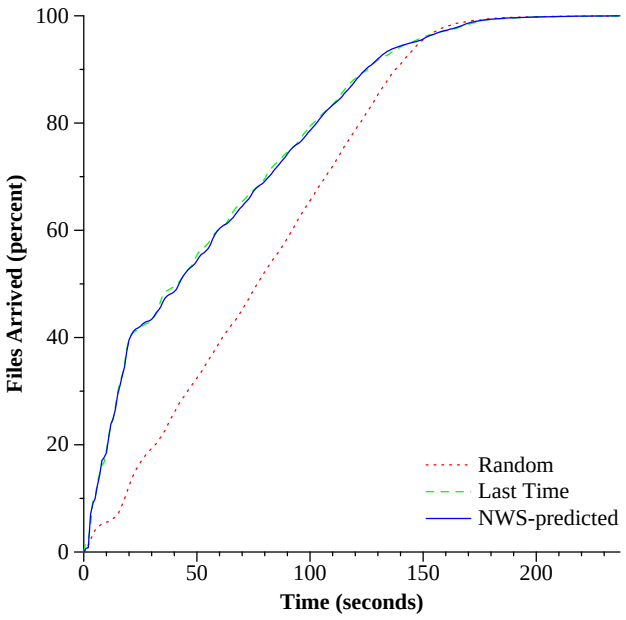

Figure 15: Livny Aggregation CDF (4 parallel downloads)

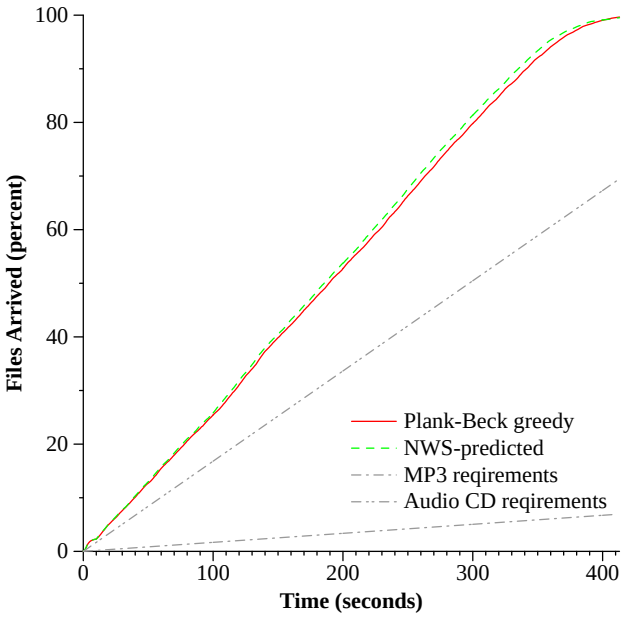

Figure 17: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (2 replicas)

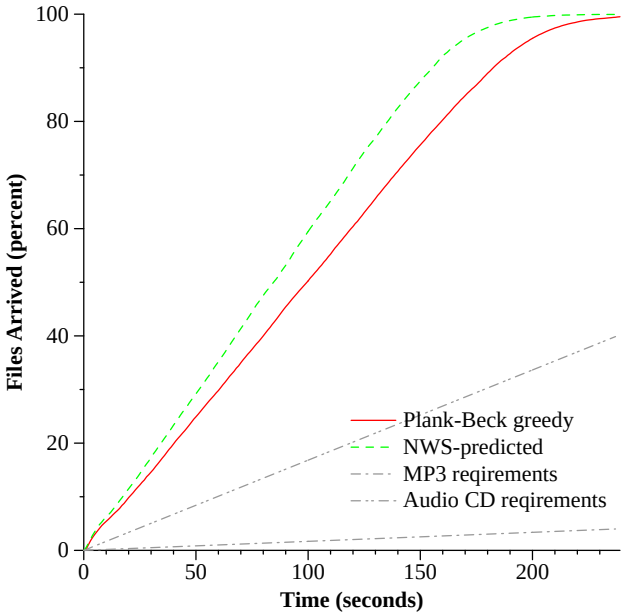

Figure 18: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (4 replicas)

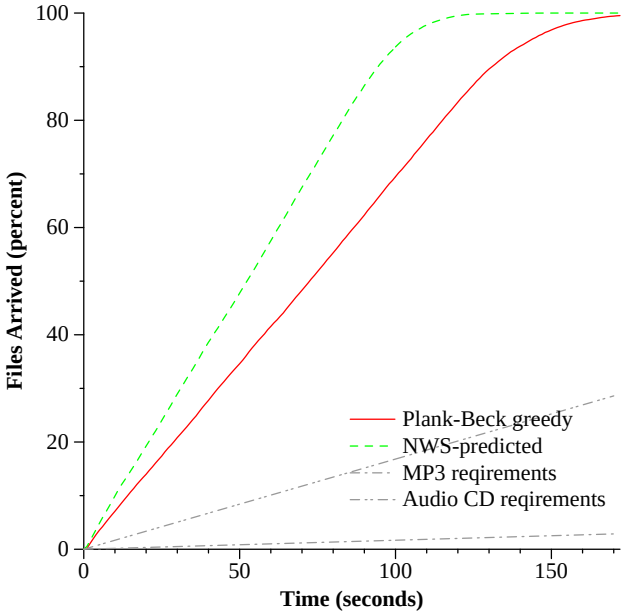

Figure 19: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (6 replicas)

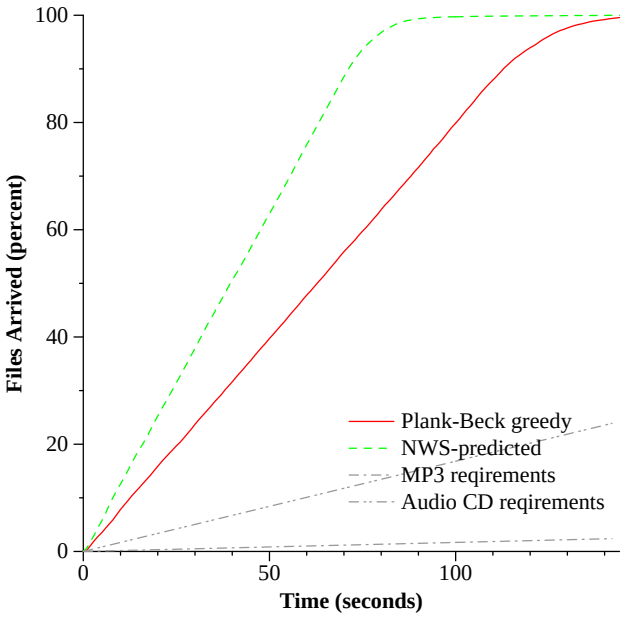

Figure 20: Plank-Beck Greedy and NWS-Predicted Aggregation CDF (8 replicas)

Plank-Beck greedy and NWS-predicted scheduling strategies for the Plank-Beck problem. Each CDF displays the percentage of the complete file that has arrived at any point in the aggregation's run time for different levels of replication. These graphs show results when this experiment was run with 2, 4, 6, and 8 replicas of each segment.

Figures 21 and *23* are duplicates of *figures 7* and *8*. *Figures 22*, *23*, *24*, and *25* graphically represent the data contained in *table 5* in the results section. These figures show the progress-driven redundancy, general progress-driven redundancy, and NWS-predicted scheduling strategies for the Plank-Beck problem. Each CDF displays the percentage of the complete file that has arrived at any point in the aggregation's run time for different levels of replication. These graphs show results when this experiment was run with 2, 3, 4, 5, and 6 replicas of each segment.

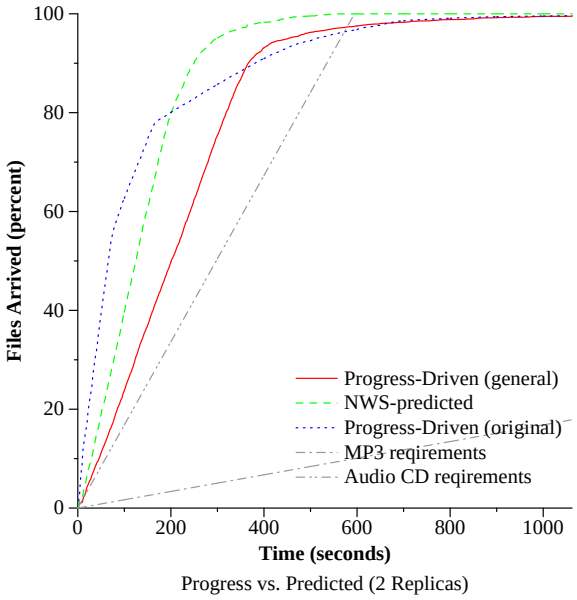

Figure 21: Progress-Driven and NWS-Predicted Aggregation CDF (2 replicas)

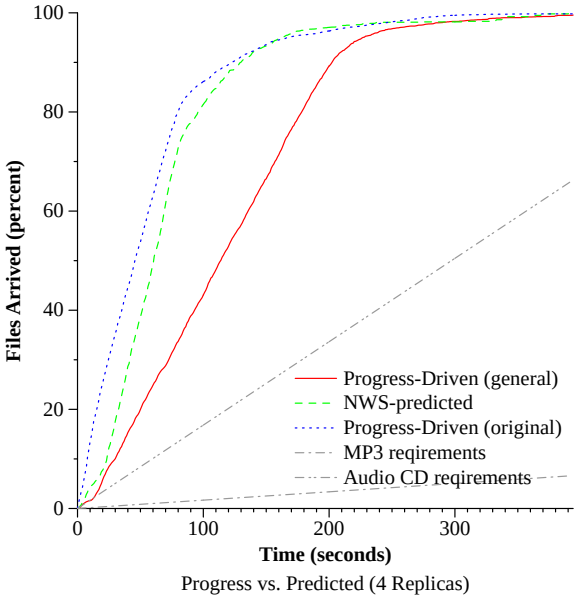

Figure 23: Progress-Driven and NWS-Predicted Aggregation CDF (4 replicas)

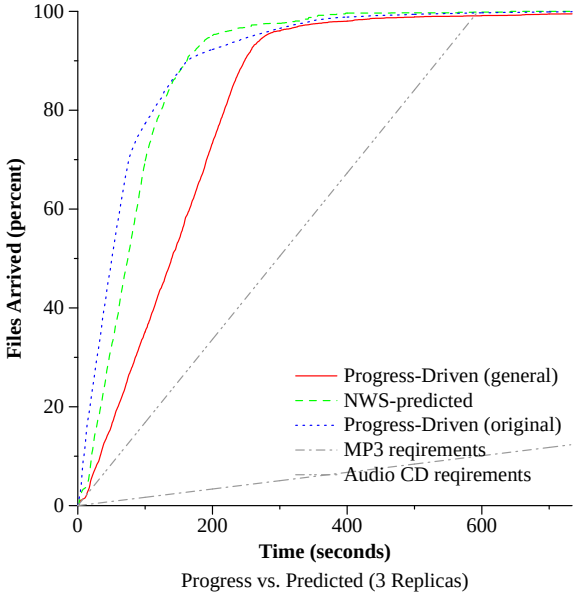

Figure 22: Progress-Driven and NWS-Predicted Aggregation CDF (3 replicas)

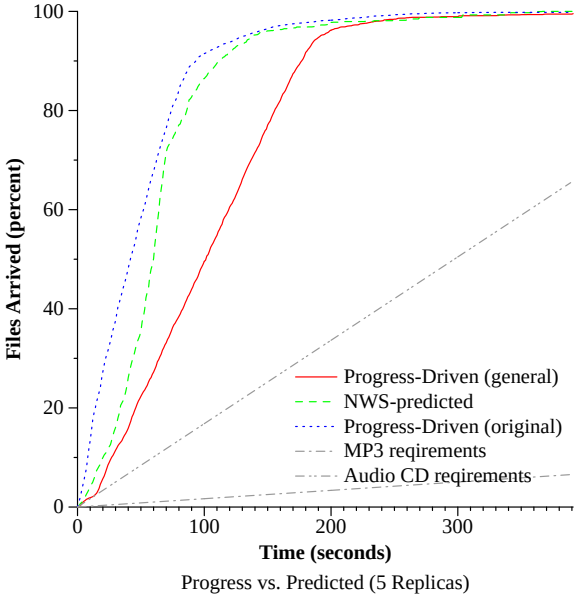

Figure 24: Progress-Driven and NWS-Predicted Aggregation CDF (5 replicas)

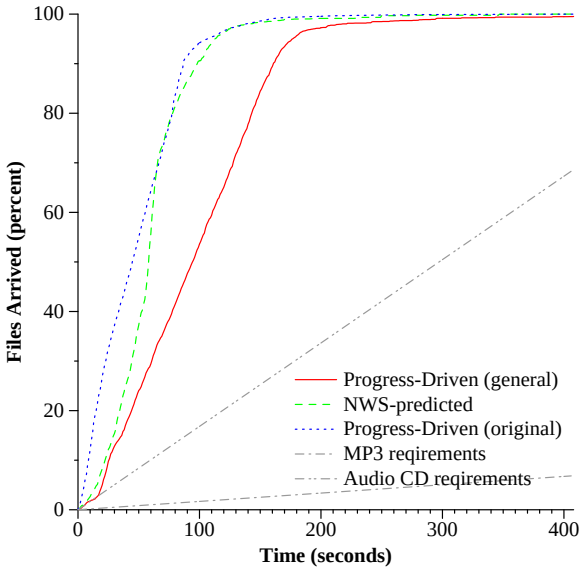

Figure 25: Progress-Driven and NWS-Predicted Aggregation CDF (6 replicas)