UNIVERSITY OF CALIFORNIA, SAN DIEGO

On-line Parallel Tomography

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science in
Computer Science

by

Shava Smallen

Committee in charge:

Professor Francine Berman, Chair
Professor Scott B. Baden
Professor Mark Ellisman

2001

The thesis of Shava Smallen is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____
Chair

University of California, San Diego

2001

To my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

Working on this thesis has been a really great learning experience for me. It has also been an enjoyable experience largely in part to all of the great people I have met and interacted with while working on this project.

First, I would like to thank my advisor, Fran Berman, for all of her support and inspiration. She has been a great role model and her guidance has allowed me to grow a lot over these past three years.

I am also extremely grateful to my co-advisor, Henri Casanova, who has been a great mentor and has been there to provide feedback and encouragement whenever I needed it.

Special thanks to Rich Wolski for his insightful comments and always clear (and entertaining) explanations.

Furthermore, I would like to thank the members of my committe, Scott Baden and Mark Ellisman.

I would also like to express my gratitude to all the folks that I have worked with on the Telescience project for which this work grew out of. I would especially like to thank Mei-Hui Su from ISI who wrote the original GTOMO code and has been incredibly wonderful to work with. From NCMIR, I would especially like to thank Steve Lamont who wrote the original tomography code and has graciously answered many tomography-related questions for me; Dave Foster and Mona Wong for providing NCMIR systems support; and Marty Hadida-Hassan.

The experiment results presented in this thesis were run in parallel using APST (AppLeS Parameter Sweep Template) developed by Henri Casanova and a number of workstation clusters. I would like to thank Satoshi Matsuoka for use of the Prospero and Presto clusters at the Tokyo Institute of Technology, Phil Papadopoulos for use of the Meteor cluster at SDSC, and David Hutches for use of the Active Web cluster at UCSD.

Additionally, I would like to thank Robert Ellis, Roummel Marcia, and Tucker McElroy from the Graduate Mathematics Consulting Group at UCSD.

Last, but definitely not least, thanks to all the folks in the Grid Computing Lab. This group has been incredibly supportive, a great source of technical information, and just fun to be around. I would especially like to thank Holly Dail and Alan Su for always being there to provide feedback. Thanks to Walfredo Cirne and Jaime Frey who worked with me on the off-line GTOMO code, Jim Hayes for providing software engineering advice, and Graziano Obertelli for administering the circus machines and supporting my laptop whenever it was in trouble. Special thanks to Marcio Faerman, Gary Shao, Otto Sievert, Renata Teixeira, and Dmitrii Zagorodnov. Also, thanks to Nadya Williams for providing chocolate support.

ABSTRACT OF THE THESIS

On-line Parallel Tomography

by

Shava Smallen
Master of Science in Computer Science
University of California, San Diego, 2001
Professor Francine Berman, Chair

Tomography is a computationally intensive process by which the three-dimensional structure of an object can be reconstructed from a series of two-dimensional projections. In this thesis, we address *on-line* execution of tomography to provide real-time feedback to users collecting data from an on-line instrument. Context for this work is provided by a powerful electron microscope located at the National Center for Microscopy and Imaging Research (NCMIR). Acquiring data from NCMIR's microscope is a lengthy process and is susceptible to configuration errors. Thus, real-time tomography feedback will allow users to quickly identify configuration problems and interact with the microscope in order to more efficiently acquire data from it.

We present an implementation of on-line parallel tomography which allows for production runs in *Computational Grid* environments. Developing applications that leverage this type of platform is difficult because resources are heterogenous and dynamic. In our approach, on-line parallel tomography is designed to be *tunable* such that it can be configured to adapt to different resource availabilities. It is coupled with an *user-directed*, *application-level* scheduler which exploits the tunability of the application to determine a schedule for *soft* real-time execution. The scheduler utilizes user constraints, an application model, and dynamic resource load predictions to determine *feasible* run-time configurations.

The configurations are displayed as choices to the user where each configuration involves trade-offs between resolution of the reconstruction, frequency of feedback, and cost of execution. Once an appropriate configuration is chosen by the user, the scheduler selects resources, allocates work, and executes the application.

Chapter I

Introduction

Reconstructing the three-dimensional structure of an object from a series of two-dimensional projections is called *tomography*. Tomography has been applied to many fields such as medical imaging, earth science, and astronomy [27]. In this thesis, we concentrate on the application of tomography to electron microscopy. Context for this work is provided by the National Center for Microscopy and Imaging Research (NCMIR) where tomography is run on data collected from an intermediate-high voltage transmission electron microscope (IVEM). NCMIR's electron microscope allows scientists to study specimens at the cellular and subcellular level and is one of the few of its kind in the United States that is available to the biological research community [25]. During a session with the electron microscope, a specimen is rotated about a single axis while projections are acquired from a CCD (charge-coupled device) camera. Typically 61 projections are acquired, where the size of each projection depends on the resolution of the CCD camera, currently either $1k \times 1k$ or $2k \times 2k$. In Figure I.1, we show an example of a tomographic volume generated from a spiny dendrite data set that was collected from NCMIR's electron microscope.

The tomographic algorithms used by NCMIR are computationally intensive. They include the R-weighted backprojection algorithm which performs the tomographic reconstruction; it is optionally followed by iterative ART (Algebraic

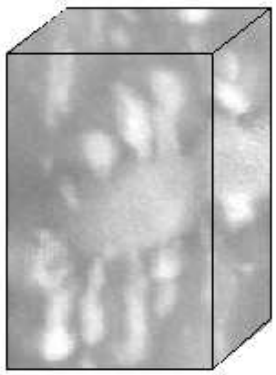


Figure I.1: Spiny dendrite.

Reconstruction Technique) or SIRT (Simultaneous Iterative Reconstruction Technique) algorithms which further refine the volume [38]. Fortunately these algorithms are also embarrassingly parallel which facilitates a parallel implementation of tomography [39]. Figure I.2 illustrates the parallelism of these tomographic algorithms. The information required to produce the ith X-Z slice of the volume is the ith scanline from all projections. Therefore, the three-dimensional volume can be decomposed into a series of X-Z slices where each slice is computed independently of the others.

There are two scenarios for which NCMIR is interested in using parallel tomography: *off-line* parallel tomography and *on-line* parallel tomography. In off-line parallel tomography, a user is interested in running tomography on a dataset that resides somewhere on secondary storage. The user's goal is to obtain a single, high-resolution tomogram as soon as possible. Conversely, in on-line parallel tomography, a user is interested in running tomography on data as it is collected from the microscope. The user's goal is to compute successive tomograms in quasi-real-time in order to obtain feedback on the quality of the data acquisition.

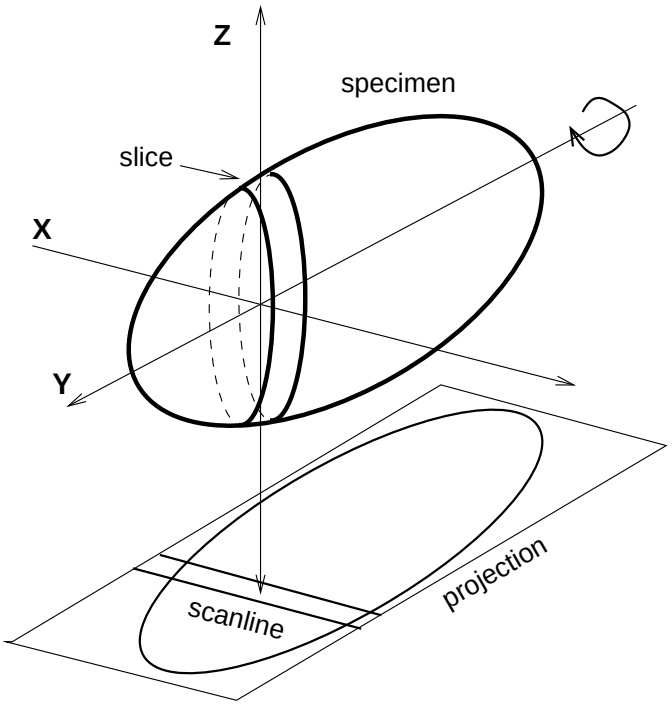


Figure I.2: Parallelism of tomography (adapted from [21]). The information required to reconstruct the ith X-Z slice is the ith scanline from all projections.

I.A Off-line Parallel Tomography

Traditionally, NCMIR scientists have run parallel tomography on data sets previously collected from the electron microscope. This procedure is referred to as *off-line* parallel tomography and is illustrated in Figure I.3. A data set of p projections, each of size $x \times y$, is acquired from the microscope and then preprocessed to correct for imperfections of the data acquisition process (e.g. fiducial alignment, normalization) [39]. Next, the projections are transformed into y *sinograms* of dimension $x \times p$, where the ith sinogram is composed from the ith scanlines of each projection. The sinograms will then be parallel processed into *slices* of the tomogram. A slice is of dimension $x \times z$, where the value for z is derived from the actual physical thickness of the specimen in pixels. Finally, the slices are collected into a *tomogram*, a three-dimensional volume, and viewed by the user.

We define an off-line parallel tomography experiment, E_{off}, using the parameters that determine the amount of data and computation involved in the tomographic reconstruction.

Definition I.1

$$E_{off} = (p, x, y, z)$$

where

- *p is the total number of projections acquired from the microscope,*
- *x is the width of the projection,*
- *y is the height of the projection (also the number of slices to compute), and*
- *z is the thickness of the specimen.*

Given NCMIR's $1k \times 1k$ and $2k \times 2k$ CCD cameras, the following are representative examples of the size of experiments run by NCMIR users: (61, 1024, 1024, 300) and (61, 2048, 2048, 600).

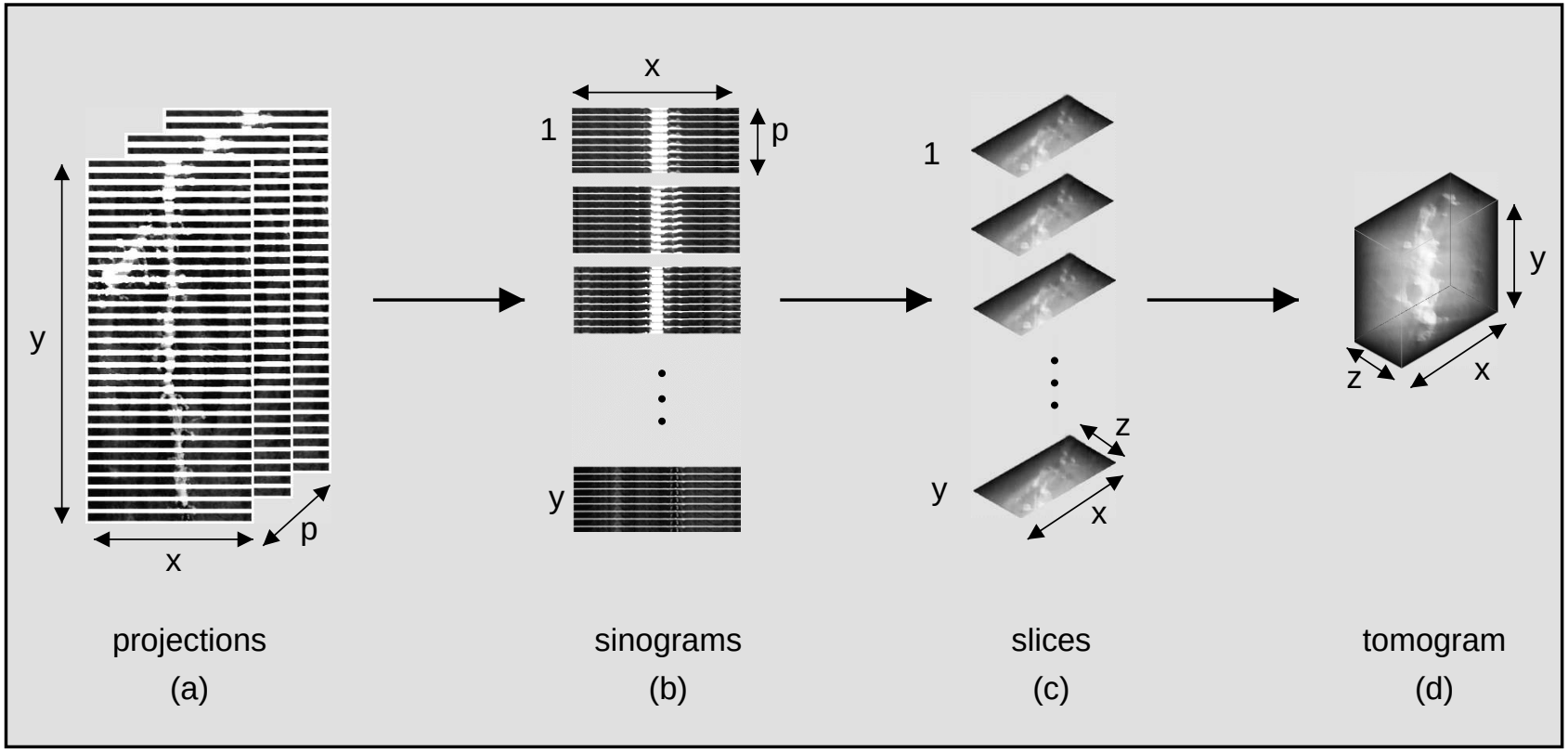


Figure I.3: Processing steps of tomography.

I.A.1 GTOMO

We and our collaborators implemented a version of off-line parallel tomography called GTOMO [46] which is targeted to a *Computational Grid* [18]. Traditionally, it can be challenging to develop applications that leverage this type of platform because resources are heterogeneous, dynamic, and governed by different administrative policies (i.e., accounting, local scheduler, security, etc.). Fortunately, there are several Grid infrastructure projects [19, 24, 30, 6, 43] available to facilitate running an application across different administrative domains. In GTOMO, we use services from the Globus toolkit [19] for remote process control and interprocess communication. We then implemented a scheduler for running off-line parallel tomography in a heterogeneous, dynamic environment.

Since the tomographic algorithms used by NCMIR are embarrassingly parallel, we can employ a *self-scheduling* [26] strategy. In GTOMO, we use a simple *work queue* algorithm where one slice of work is assigned to a processor at a time until all slices have been processed. However, effective resource selection is more complicated because NCMIR's platform includes space-shared resources (supercomputers). On space-shared resources, jobs execute on dedicated processors but typically have to wait in a queue before execution. Depending on the number of processors requested and the load of the machine, the queue time of a job can range from seconds to days. In GTOMO, we implemented a coallocation strategy that avoids queue time delays entirely by adaptively submitting job requests that can start running immediately. To submit a job request that starts immediately, we utilize availability information exported from a supercomputer's batch scheduler such as the Maui Scheduler [33]; this information includes the number of nodes available for immediate use and, more importantly, the length of time for which they are available. Therefore, we say our strategy coallocates the execution of parallel tomography over workstations and *immediately available supercomputer processors*. The coallocation strategy implemented in GTOMO was implemented as an AppLeS. An AppLeS (application-level scheduler) [1] integrates with the

target application to develop a schedule for deploying the application in a Grid environment. The scheduler makes predictions of the performance the application may experience on prospective resources at execution time. Using these predictions, a potentially performance-efficient schedule for the application is identified and deployed [49, 48, 16, 46]. In [46], we showed that the GTOMO AppLeS strategy improved the turnaround time of off-line parallel tomography over strategies that targeted either workstations or supercomputers alone. Currently, GTOMO is used in production at NCMIR on multi-user workstation clusters and supercomputers.

The architecture of GTOMO is displayed in Figure I.4. There are four types of processes in GTOMO: driver, reader, writer, and ptomo. The *driver* is invoked by the user and starts up all other processes using the AppLeS coallocation strategy. It also coordinates interactions among the different processes and controls the work queue. The *reader* and *writer* are multi-threaded I/O processes and have direct access to the user's file system. The reader reads input files off the disk and sends them to the ptomos for processing. The writer receives output files from ptomos and writes them to disk. Note that the reader and writer enable GTOMO to run across different file systems. The *ptomo* receives input files from a reader, does all the computational work, and sends the output to a writer. Since data is typically read and written to one disk (not necessarily the same disk), we use one reader, one writer, and any number of ptomos.

I.B On-line Parallel Tomography

The time to acquire a single projection from NCMIR's electron microscope ranges from 45 seconds to 3 minutes. Therefore, it can take at least 45 minutes to acquire a complete data set of 61 projections. When the user visualizes the data at the end of the acquisition process, they might discover that the data is flawed or might find a better area of the specimen to study. In this case, the user will restart the whole experiment with different parameters. It would therefore

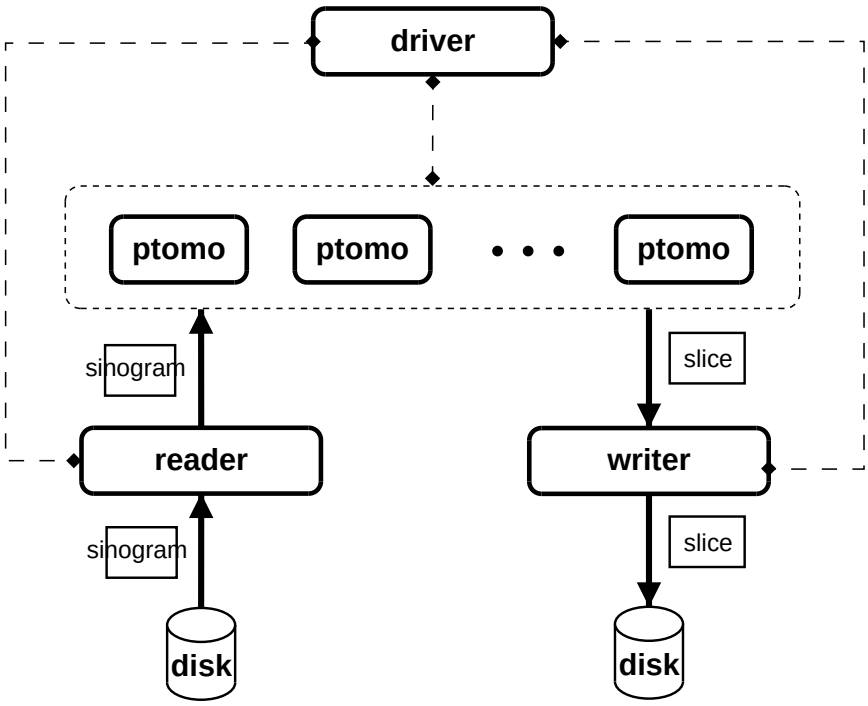


Figure I.4: Architecture of GTOMO, the off-line parallel tomography implementation.

be useful to compute tomograms during data acquisition to provide users with feedback on the quality of their data; each successive tomogram would reveal more information about the three-dimensional structure of the specimen. This would allow for more efficient use of the microscope because users would be able to make changes to their experiment early during the acquisition process. Furthermore, it could potentially reduce the amount of specimen damage by limiting exposure to the electron beam [47].

The procedure of on-line parallel tomography is as follows: When the first projection is collected from the microscope, a coarse tomogram of the specimen is generated. Each projection is then successively processed in order to refine the tomogram with additional data. We define the *acquisition period* as the time to acquire a projection from NCMIR's electron microscope. NCMIR is currently targeting an acquisition period of 45 seconds; therefore, we use this value throughout this thesis.

We define an on-line parallel tomography experiment, E_{on}, using a set of parameters which describe the data collected from the electron microscope.

Definition I.2

$$E_{on} = (a, p, x, y, z)$$

and

- *a is the time to acquire a projection from the microscope,*
- *p is the total number of projections acquired from the microscope,*
- *x is the width of the projection,*
- *y is the height of the projection (also the number of slices to compute), and*
- *z is the thickness of the specimen.*

Note that the refinement process involves changing the values of pixels within slices of the tomogram. Therefore, the size of the tomogram is constant throughout data

acquisition.

I.C Thesis Summary

In this thesis, we describe an extension to GTOMO to support on-line parallel tomography. Because on-line parallel tomography is resource-intensive and our target platform is dynamic, we implemented on-line parallel tomography as a *tunable* application. A tunable application is characterized by the availability of alternate configurations, where each configuration corresponds to a different execution path and resource usage [9]. For on-line parallel tomography, a configuration is defined by the resolution of the tomogram, frequency of refinements to the tomogram, and cost of execution. These parameters allow configuration of on-line parallel tomography to accommodate different resource availabilities.

Second, we describe the implementation of an *user-directed* AppLeS that exploits the tunability of on-line parallel tomography in order to adaptively schedule its execution onto a set of resources. The AppLeS is implemented as multiple constrained optimization problems derived from an application model, user information, and dynamic resource load information. This methodology is flexible and can be solved efficiently using linear programming.

I.D Organization of Thesis

In Chapter II, we discuss the motivation and implementation of on-line parallel tomography as a tunable application. Chapter III details the design and implementation of the user-directed AppLeS. Three sets of experimental results are described in Chapter IV. The first set of experiments described in Section IV.B shows that dynamic resource load information, in particular bandwidth information, is key to real-time execution performance. In Section IV.C, we show that tunability is an important application characteristic for running on-line parallel

tomography in a multi-user, dynamic environment. Finally, in Section IV.D we evaluate the scheduling latency introduced by the AppLeS. We discuss related work in Chapter V and conclude the thesis in Chapter VI.

Chapter II

Tunable On-line Parallel Tomography

As discussed in Chapter I, we have implemented on-line parallel tomography as a tunable application; i.e., an application whose configuration is determined by a set of parameters which can be varied or "tuned". Tunability is an important application characteristic for running on-line parallel tomography in a dynamic Grid environment since resource availability changes over time. Tuning parameters allow the application to be configured to adapt to run-time resource availability. In Section II.A, we discuss the motivation and implementation of the GTOMO extension to allow for tunable on-line parallel tomography. In Section II.B, we discuss the three parameters that define a configuration of on-line parallel tomography: resolution of the tomogram, frequency of refinements to the tomogram, and cost. s

II.A GTOMO Extension

To motivate the required changes to GTOMO to allow for on-line parallel tomography, we first discuss how the current off-line GTOMO design is insufficient for on-line parallel tomography. Suppose that a NCMIR user wants to run an

on-line parallel tomography experiment $E = (45, 61, 2048, 2048, 600)$ as described in Definition I.2 and Section I.A. If NCMIR had access to resources of infinite capability (i.e., infinite bandwidth links and infinite processor speed), we would be able to run the off-line implementation of parallel tomography after each projection was acquired from the microscope and have it complete instantaneously. Thus, users would be able to obtain the highest resolution tomogram possible and would see refinements of the tomogram at the highest frequency possible, the microscope acquisition rate. Now, let us consider E for a set of more realistic resources.

Using Definition I.2, there will be 2048 slices of work to process for experiment E. To process a single scanline of a projection into a slice using the R-weighted backprojection method [41] takes approximately .33 seconds on a dedicated 700 MHz AMD Athlon processor (see Figure II.1 for a description of the R-weighted backproject algorithm). Using this as an average processor speed, the first refinement of the tomogram (or *refresh*) would take $.33 \times 2048 \approx 676$ seconds. Under the current implementation of GTOMO, each successive tomogram refresh computation would repeat the work done to compute the previous tomogram refresh. This is due to GTOMO's scheduling strategy (work queue); a ptomo processes one slice of work at a time, sends it to the writer, and then deletes it (i.e., a ptomo is stateless). Therefore, when a new projection is acquired from the microscope, all data must be sent out again and processed. Consequently, the second refresh of the tomogram would take $2 \times .33 \times 2048 \approx 1352$ seconds since ptomo must reprocess the scanline from the previous projection and then process the scanline from the new projection. Likewise, the third refresh would take $3 \times .33 \times 2048 \approx 2028$ second; the last refresh would take $61 \times .33 \times 2048 \approx 41,226$ seconds. To execute in real-time, we want the processing of one projection to complete before the next one arrives. Assuming optimal parallelization speedup, the first refresh would require $\lceil 676/45 \rceil = 16$ processors, the second refresh would require $\lceil 1352/45 \rceil = 31$ processors, and the last refresh would require $\lceil 41,226/45 \rceil = 917$ processors. This technique requires an increas-

```
Algorithm : BACKPROJECTSCANLINE(scanline, slice, angle)

local height, width

height ← getSliceHeight(slice)
width ← getSliceWidth(slice)

RWeightScanline(scanline)
for y ← 0 to height − 1
   for x ← 0 to width − 1
      slice[y][x] ← slice[y][x] + calculateContribution(angle, scanline)
```

Figure II.1: Algorithm for backprojecting a single *scanline* of a projection (at *angle*) into a *slice* of the volume. First, the scanline is modified using the *RWeightScanline* function to smooth the data. Then, every pixel of the slice is updated to consider the contribution of the scanline.

ing amount of computational power; furthermore it is inefficient because it is not augmentable. To be *augmentable*, a technique should allow each successive computation to build upon the previous computation without repeating work. Hence, a more efficient technique would be to store all previous computation so that refreshes do not repeat work. Therefore, we added an extension to GTOMO so that the R-weighted backprojection algorithm can be executed as an augmentable technique.

Our approach is to use a static work allocation strategy. A *static work allocation* is a fixed assignment of computation to a set of resources. In this context, a static work allocation is an assignment of y slices of work among a set of ptomos. We then modify the ptomos so that they are stateful. In particular, whenever a projection is acquired from the microscope, the ith scanline is sent to the ptomo that has been allocated the ith slice so that it may process the new data. The advantage of this technique is that we reduce the computation by a factor $\sum_{i=1}^{p} i$, where p is the total number of projections acquired from the microscope. Therefore, in the example presented in the previous paragraph, each

refresh would require 672 seconds since we only process the scanline data from the new projection for each refresh. Therefore, we need 16 processors for the entire computation. The drawback of this approach is that we use lose the run-time adaptive scheduling advantage of work queue [26] used in the off-line GTOMO case; we address this tradeoff in Chapter VI.

The structure of GTOMO on-line parallel tomography extension is shown in Figure II.2. As in the off-line parallel tomography mode, the *driver* is invoked by the user and starts up all other processes. The *electron microscope* sends a projection to the *preprocessor* every a seconds. The preprocessor divides the projection into sections, where each section contains multiple scanlines. The sections are allocated to *ptomo* processes such that the scanlines in each section can be processed in parallel. All ptomos will periodically send their slices to the *writer* in order to update the tomogram. A visualization program will then display updated tomograms to the user.

As a final note, recall from Chapter I that the optional iterative ART and SIRT algorithms operate on the specimen data after the R-weighted backprojection completes. In each iteration, the tomogram is corrected based on differences between the original projection data and reconstructed volume. For the ART [23] algorithm, pixels in the slices are corrected p times during a single iteration; the correction for a pixel (from the ith slice) is calculated from the ith scanline from one of the projections. For the SIRT [22] algorithm, each pixel in the ith slice gets updated once during a single iteration using a correction based on the ith scanlines from all projections (also known as a sinogram). Therefore, since both the ART and SIRT algorithms assume all data has been acquired from the microscope (each iteration involves data from all projections), these algorithms are not augmentable. Hence, for on-line parallel tomography we only use the R-weighted backprojection which computes sufficiently refined tomographic reconstructions to provide feedback on the quality of the data acquisition. Note that if the user wants to refine their tomogram with the ART or SIRT algorithms, they can run GTOMO

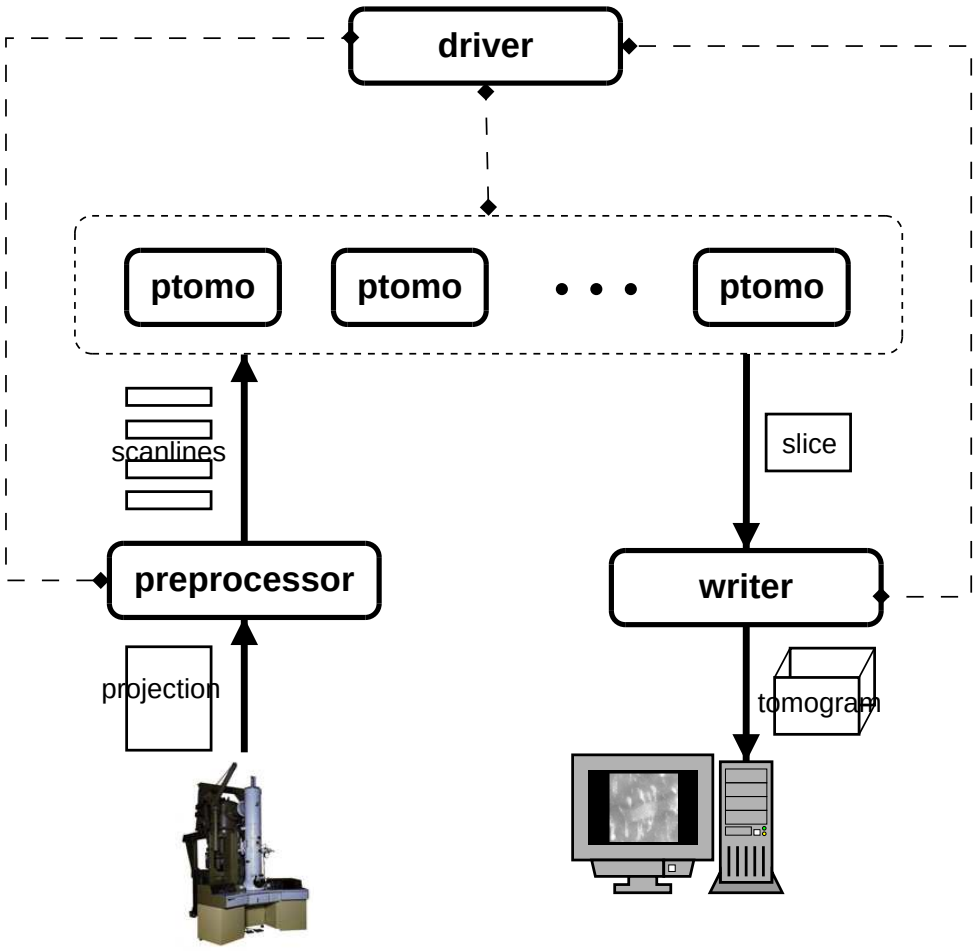


Figure II.2: Architecture of GTOMO on-line parallel tomography extension. The electron microscope sends data to the preprocessor. The data is then allocated to the ptomos to be processed in parallel. The output data is collected by a writer process where it can be visualized.

in off-line parallel tomography mode after data acquisition is complete.

II.B Tunable Parameters

We now define the parameters that define a configuration of on-line parallel tomography. These parameters will allow the application to be tuned to adapt to different resource availabilities.

Consider the communication associated with the experiment E = (45, 61, 2048, 2048, 600). Each slice will be about 4.7 MB, yielding a tomogram of 9.6 GB. If we place our writer on a machine with an observable bandwidth of 300 Mb/s, it will take 1024 seconds (17 minutes) to transfer the whole tomogram. Note that since ptomos prefetch slices into memory using multi-threading, we neglect disk access time. Given that we do not want to overload the network by sending a tomogram before the transfer of the previous tomogram has completed, we can send a refined tomogram to the writer every $\lceil 1024/45 \rceil = 23$ projections. We therefore say the number of *projections per refresh* is 23 and the *refresh period* is $23 \times 45 = 1035$ seconds (17.25 minutes). Since NCMIR users would like refreshes to complete within 10 minutes, this is unacceptable. One solution is to *reduce* the size of the projections. Suppose we reduce the projections by a factor of 2 in each dimension.[1] We will then have an experiment E' = (45, 1024, 1024, 300) to process. Therefore, each slice will be about 1.2 MB, yielding a tomogram of 1.2 GB, 8 times smaller than the $2k \times 2k$ data set. If we again assume 300 Mb/s bandwidth, it will take 128 seconds to transfer each tomogram which would reduce the number of projections per refresh to 3. Similarly, if we were to reduce by a factor of 4, it would take 16 seconds to transfer each tomogram which would reduce the projections per refresh to 1, the best refresh frequency possible. Given that we cannot predict what trade-offs will be preferable to a user, we let each user

[1]Note that it takes about 1.3 seconds to reduce a $2k \times 2k$ projection on a 700 MHz AMD Athlon processor. Therefore, we introduce a latency of 1.3 seconds in the time to acquire the initial projection from the microscope. However, the period between successive projections will not be affected; therefore, the acquisition period will also not be affected.

individually decide which configuration is best for them.

Note that the communication associated with the input data is relatively small compared with that of the output data. For example, in a $2k \times 2k$ data set, projections are 16 MB, whereas a tomogram is 9.6 GB. For a $1k \times 1k$ experiment, each projection would then be only 4 MB whereas each tomogram would be 1.2 GB. In both cases, the output data is two orders of magnitude larger than the input data set.

We now formally define two parameters that determine the quality of an execution of on-line parallel tomography: *reduction factor* (f) and *projections per refresh* (r). We then define a third cost parameter, *service units* (su). The configuration of on-line parallel tomography is defined by a *triple* (f, r, su). We describe each of these in more detail below.

II.B.1 Reduction Factor

The *reduction factor* (f) is a scalar integer value that results in a reduction of the size of a projection in each dimension. For example, if we reduce a projection of size $x \times y$ by f, we will have a projection of size $\frac{x}{f} \times \frac{y}{f}$. An increase in the reduction factor decreases both the number of slices to compute and the amount of computation per slice. For the time being, we consider just a simple averaging reduction method. We modified the averaging algorithm given in [28] so that it works for arbitrary reduction factors. The modified averaging algorithm, given in Figure II.3, works by first dividing the $x \times y$ projection into square *windows* of size $f \times f$. For each window, the values of the pixels are averaged to create a single pixel in the reduced projection. Figure II.4 shows an 8×8 projection reduced by a factor of 2. Note that in order to yield a sufficiently detailed tomogram for NCMIR users, projections should not be reduced beyond 256×256. For example, the maximum f for a $1k \times 1k$ experiment is 4 and the maximum f for a $2k \times 2k$ experiment is 8.

Algorithm : REDUCE$(projection, x, y, f)$

```
for i ← 1 to y/f
   for j ← 1 to x/f
      sum ← 0
      for m ← 1 to f
         for n ← 1 to f
            sum = sum + projection[i * f + m][j * f + n]
      reducedProjection[i][j] = sum/(f * f)
return (reducedProjection)
```

Figure II.3: Reduction algorithm.

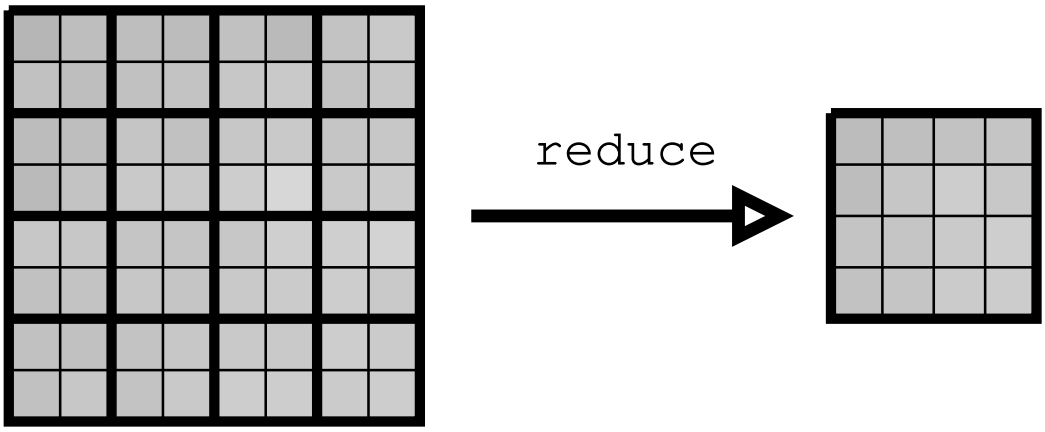


Figure II.4: A 8×8 projection being reduced by a factor of 2.

II.B.2 Projections Per Refresh

The *projections per refresh* (r) parameter refers to the number of new projections processed into each successive tomogram refinement or *refresh*. For example, if $r = 3$, a user would see a refreshed tomogram after every third projection was acquired from the microscope. We refer to the time to complete a refresh as the *refresh period*. The refresh period can be determined by multiplying r by the acquisition period, a. Increasing r reduces the frequency of refreshes sent to the user and thus reduces the amount of communication. As mentioned previously, an upper bound on the time between successive tomogram refresh is 10 minutes for NCMIR users. Therefore, for an acquisition period of 45 seconds, r should be no more than $\lfloor 600/45 \rfloor = 13$.

II.B.3 Cost

Thus far, we have assumed that all resources are free. While this model may be appropriate for workstations where resource usage is not monitored, it is not appropriate for many supercomputers. At supercomputer centers, resource usage is generally monitored through allocation [13, 10, 35, 37, 34]. Usually a research group is given an allocation of supercomputer time per quarter. If the group exceeds their allocation, they will no longer be allowed to run on that resource for the duration of that quarter. Therefore, a group may want to moderate their supercomputer usage. We define a parameter, *service units* (su), for on-line parallel tomography to indicate how much supercomputer time will be consumed by a run. Service units are calculated using the following equation based on the wall clock charging policies of five supercomputer centers [13, 10, 35, 37, 34].

$$su = charge\ factor \times number\ of\ CPUs \times wall\ clock\ time \qquad \text{(II.1)}$$

The *charge factor* is simply a generic integer value to account for different charging policies enforced by supercomputer centers. The charge factor could be based on

the type of user, the queue type, or some other factor specific to the supercomputer center.

II.C Summary

In this chapter, we described an extension to GTOMO to allow for on-line parallel tomography. The extension enables the R-weighted backprojection method to execute as an augmentable technique. This is more efficient than running off-line parallel tomography multiple times but loses the run-time adaptive scheduling advantage of work queue. We then defined a configuration of on-line parallel tomography as a triple of tunable parameters, (f, r, su). These parameters represent resolution of the tomogram, frequency of refinements to the tomogram, and cost. As described in the next chapter, these parameters will allow the AppLeS to adapt the application configuration to the availability of a set of resources.

Chapter III

User-Directed AppLeS

In the previous chapter, we discussed the design of on-line parallel tomography as a tunable application. However, it is difficult to choose a configuration and work allocation that efficiently utilize multi-user, dynamic sets of resources at run time. First, determining an appropriate work allocation requires availability information for each resource (e.g. CPU, bandwidth). Second, since these are dynamic resources, the best configuration will vary over time. In this chapter, we discuss the design of an *user-directed* AppLeS for on-line parallel tomography. In Section III.A, we motivate and define a user-directed AppLeS. We then describe the design of the AppLeS in Section III.B and III.C.

III.A Design

In Section II.B, we defined a triple (f, r, su) that determined the configuration of on-line parallel tomography. If enough resources are available, users will always want to run using the best configuration, $(1, 1, 0)$. This would result in the highest resolution tomogram being refreshed at the highest frequency possible for zero cost. Yet, in practice, resource availability may prevent users from achieving this configuration. In this case, users will need to choose an alternate configuration. However, it is is not always obvious which configuration is the best

$f = 1$	$f = 2$	$f = 2$
$r = 6$	$r = 2$	$r = 1$
$su = 4$	$su = 8$	$su = 20$
(a)	(b)	(c)

Table III.1: Three example configurations available for a tomographic reconstruction and resource platform.

alternative.

Suppose the configurations listed in Table III.1 are three possible configurations for a tomographic reconstruction and target platform. Without some knowledge of the user's criteria, it is not obvious which configuration is the best. Furthermore, choosing a configuration that favors one parameter may involve trading off the benefits of another parameter. For example, a higher f would allow for a smaller r (since there would be less data to transfer). Also, a higher su could result in a lower f and/or lower r (since there would be more computational power). In the example presented in Table III.1, if resolution was the most important parameter, (a) would be the best choice for a user. On the other hand, if frequency of refreshes was more important, (b) or (c) would be better choices; (c) would be the best choice if spending 20 service units was acceptable.

Automating the process of determining the best configuration for a user is beyond the scope of this thesis. In this work, the AppLeS *assists* users in selecting a configuration that works for them and is thus referred to as a *user-directed* AppLeS. The design of the user-directed AppLeS scheduler is illustrated in the flow diagram shown in Figure III.1. The grayed shapes correspond to user actions while the white shapes correspond to AppLeS actions. We detail each step in the following description.

(i) The user specifies bounds on each configurable parameter: f, r, and su. This corresponds to the maximum and minimum value the user is willing to tolerate for a parameter.

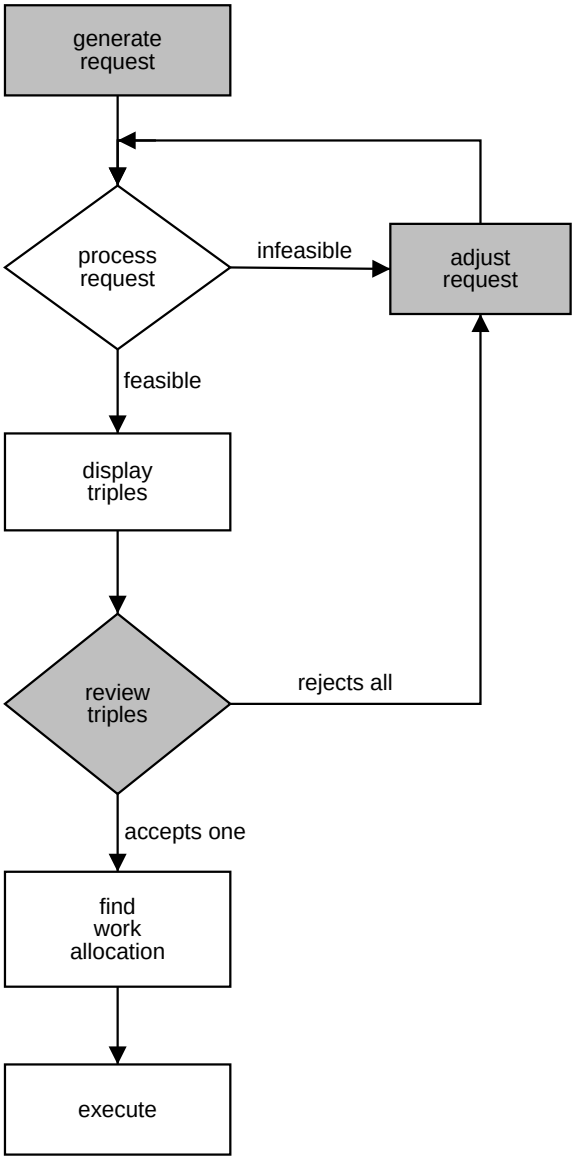


Figure III.1: Flow diagram for a user-directed AppLeS. The grayed shapes correspond to user actions while the white shapes correspond to AppLeS actions.

(ii) The AppLeS searches the parameter space for feasible triples; each triple corresponds to a feasible configuration of the tunable application. If no configurations can be found, the user will need to adjust the request.

(iii) The user considers all configurations and then selects a single triple for execution.

(iv) The AppLeS will determine an appropriate work allocation for the user-selected triple and then execute.

This approach allows the user to select the best configuration for them from the set of feasible configurations determined by the AppLeS. In the following subsections, we describe how the user-directed AppLeS finds feasible triples and determines work allocation.

III.B Searching for Triples

In order for the AppLeS to search for available configurations, the user supplies it with a lower and upper bound on each parameter; this indicates the range of values the user finds acceptable for a parameter. Therefore, we say a triple (f, r, su) is a *candidate* if,

$$\begin{aligned} f_{min} \leq f \leq f_{max} \\ r_{min} \leq r \leq r_{max} \\ su_{min} \leq su \leq su_{max} \end{aligned} \tag{III.1}$$

For an experiment, E, and a set of resources, M, we say a candidate triple is *feasible* if there exists a *work allocation*, W, for it (see Section III.C); if no W can be found, we say the triple is *infeasible*.

As discussed in the previous section, our goal is to present the user with a set of *feasible* triples (f, r, su). One approach is exhaustive search. For each triple (f, r, su), one can search for a possible work allocation. A more efficient approach is to solve three optimization problems:

(i) fix f and r, minimize su;

(ii) fix r and su, minimize f; and

(iii) fix f and su, minimize r.

This approach has the added advantage of filtering out suboptimal triples. For example, suppose that triples $(1, 1, 0)$ and $(1, 2, 0)$ are feasible. We assume that users would alway choose $(1, 1, 0)$ over $(1, 2, 0)$.

We display the AppLeS search algorithm in Figure III.2. There are three loops that correspond to the three optimization problems outlined above. The three functions $findOptimalServiceUnits$, $findOptimalProjectionsPerRefresh$, and $findOptimalReductionFactor$ search for a work allocation given two fixed input parameters; this is accomplished by solving a constrained optimization problem as described in the next section. If a work allocation is found, the optimized parameter is returned and the triple is added to a list. For added efficiency, we stop searching whenever the optimized parameter found stops improving. Since the three loops may result in duplicate triples, we add a procedure at the end to remove duplicates from the list.

III.C Work Allocation Experiments

Consider an experiment $E = (a, p, x, y, z)$. The goal is to find a work allocation for a set of resources, M. We define a *work allocation* as a set W:

$$W = \{w_m : m \in M\} \tag{III.2}$$

Algorithm : SEARCH$(f_{min}, f_{max}, r_{min}, r_{max}, su_{min}, su_{max})$

```
triples ← ∅
for i ← f_min to f_max
   optimal_su ← ∞
   for j ← r_min to r_max
      if findOptimalServiceUnits(i, j, &su) == FOUND
         if su < optimal_su
            triples.add(i, j, su)
          else
            break
for i ← f_min to f_max
   optimal_r ← ∞
   for j ← su_min to su_max
      if findOptimalProjectionsPerRefresh(i, j, &r) == FOUND
         if r < optimal_r
            triples.add(i, r, j)
          else
            break
for i ← r_min to r_max
   optimal_f ← ∞
   for j ← su_min to su_max
      if findOptimalReductionFactor(i, j, &f) == FOUND
         if f < optimal_f
            triples.add(f, i, j)
          else
            break
triples.removeDuplicates()
return (triples)
```

Figure III.2: AppLeS triple search algorithm.

where w_m is the number of tomogram slices allocated to resource m. We have the following two constraints:

$$\forall m \in M \quad w_m \geq 0 \tag{III.3}$$

$$\sum_{m \in M} w_m = y. \tag{III.4}$$

Recall that there are a total of y tomogram slices to compute, i.e., we assume that there is no work replication. To find W, we first create a model of the application; the model is simply a system of equalities and inequalities. We then plug dynamic resource load information into the model and solve the system using the method described in Section III.C.5.

III.C.1 Application Model

The model for on-line parallel tomography frames it as a *soft real-time* application. A soft real-time application is characterized by the execution of tasks which have soft deadlines [31]. That is, the usefulness of a task with a soft deadline decreases as the lateness of the task increases [31, 4]. Given the discussion in Section II, our soft-deadlines are:

(i) The computation time of one projection will be less than the acquisition period.

(ii) The transfer time of a tomogram will be less than the refresh period.

If one of these deadlines is missed, performance degrades. Therefore, our goal is to find a work allocation for which all deadlines are met. We express the problem as a constrained optimization problem. In Sections III.C.2 and III.C.3 we translate the deadlines expressed above into inequalities. In Section III.C.4, we add in a set of equalities to express the cost of execution. Finally, we add in the user's bounds defined in Equation III.1. The complete system of equalities and inequalities is displayed in Figure III.7.

III.C.2 Computation

In order to satisfy the soft computation deadline outlined above, we introduce the following inequality into our model:

$$\forall m \in M \quad T_{comp}(m) \leq a, \tag{III.5}$$

where $T_{comp}(m)$ is the time to compute w_m slices on resource m and a is the acquisition period. In other words, we want the computation of one projection to complete before the next projection is acquired. Otherwise, the projections will queue up and we will lose real-time execution (i.e., refreshes to the tomogram). To determine $T_{comp}(m)$, we examine the ptomo algorithm displayed in Figure III.3. Suppose a resource m is assigned w_m slices α to β. Each time a projection is acquired from the microscope, the preprocessor will send it scanlines α to β for processing. Resource m will receive the w_m scanlines and then backproject each scanline into its appropriate slice. The execution time, t_b, for *backprojectScanline* is approximately proportional to the number of pixels in the slice (see Figure II.1). That is,

$$t_b \approx tpp_m \times \frac{x}{f} \times \frac{z}{f}, \tag{III.6}$$

where tpp_m (time per pixel) is the time in seconds to process a scanline into a single pixel of the slice on a dedicated processor of m and f is the reduction factor. Since the computation time is dominated by *backproject*, the time to compute w_m slices on a dedicated processor of m is

$$T_{comp}(m) \approx tpp_m \times \frac{x}{f} \times \frac{z}{f} \times w_m. \tag{III.7}$$

Recall that our set of resources, M, contains two types of compute resources: time-shared resources (workstations) and space-shared resources (super-

Algorithm : PROCESS(α, β)

global $projectionsPerRefresh, angleList$
local $angle, scanlines, slices$

for $projectionId \leftarrow 0$ **to** $p - 1$
  $angle \leftarrow angleList[projectionId]$
  $scanlines \leftarrow recvScanlines(\alpha, \beta)$
  for $i \leftarrow \alpha$ **to** β
    $backprojectScanline(scanlines[i], slices[i], angle)$
    if $(projectionId$ **mod** $projectionsPerRefresh) = 0$
      $sendSlice(slices[i])$

Figure III.3: Ptomo processing algorithm.

computers). Let TSR be the set of time-shared resources and SSR be the set of space-shared resources such that

$$TSR \cup SSR = M. \tag{III.8}$$

On a time-shared resource,

$$T_{comp}(m) \approx \frac{tpp_m}{cpu_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m, \tag{III.9}$$

where cpu_m is the fraction of CPU available on m. In practice, we obtain a prediction of the value for cpu_m from the Network Weather Service (NWS) [55, 16]. The NWS is a resource monitoring system that provides dynamic resource load forecasts (e.g. available CPU, bandwidth, and memory). Likewise, for a space-shared supercomputer,

$$T_{comp}(m) \approx \frac{tpp_m}{u_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m, \tag{III.10}$$

where u_m is the number of processors on m that are unused (i.e., processors immediately available for execution). We can obtain u_m from batch schedulers such as the Maui Scheduler [33] as discussed in Section I.A.1 using the command `showbf`.

In summary,

$$T_{comp}(m) \approx \begin{cases} \frac{tpp_m}{cpu_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m & \text{if } m \in TSR \\ \frac{tpp_m}{u_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m & \text{if } m \in SSR \end{cases} \tag{III.11}$$

III.C.3 Communication

For communication, we introduce the following transfer constraint into our model:

$$\forall m \in M \quad T_{comm}(m) \leq r \times a, \tag{III.12}$$

where $T_{comm}(m)$ is the time in seconds for resource m to transfer w_m slices to the writer, r is the projections per refresh, and a is the time to acquire a projection from the microscope. In other words, we want the transfer of a tomogram to complete within the refresh period.

We model the transfer time, $T_{comm}(m)$, using the equation given in [14],

$$T_{comm}(m) = T_o + \frac{c}{B_m}, \tag{III.13}$$

where T_o is the message overhead, c is the amount of data transferred, and B_m (b/s) is the transfer rate from resource m to the writer. However, since slices are generally megabytes in size (e.g. 1.2 MB, 4.7 MB), we treat T_o as a nominal value. Therefore, we say

$$T_{comm}(m) \approx \frac{c}{B_m}. \tag{III.14}$$

Given w_m slices of size $x \times z$,

$$c = w_m \times (\frac{x}{f} \times \frac{z}{f} \times sz) \tag{III.15}$$

where sz is the number of bits used to represent a pixel. In our current implementation, a pixel is stored as a float (e.g. 32 bits). We obtain a prediction on the value of B_m (b/s) from the NWS [56, 54]. Therefore,

$$T_{comm}(m) \approx \frac{w_m \times (\frac{x}{f} \times \frac{z}{f} \times sz)}{B_m}. \tag{III.16}$$

Note that this model assumes a fully connected network such as that displayed in Figure III.4. However, in practice, many resources are connected by way of a shared network link [51, 40] . For example, Figure III.5 shows a 10 Mb/s ethernet subnet and a 100 Mb/s ethernet subnet connected via a switch. Using our current model, the AppLeS would schedule as if both **A** and **B** had a bandwidth of 10 Mb/s to the writer even though they actually share the 10 Mb/s bandwidth. Therefore, we incorporate network topology information into our model in order to determine a more effective work allocation. We group resources into *subnets*, where a subnet contains a set of compute resources which share a network link to the writer. Let S be the set of subnets such that

$$\bigcup_{S_i \in S} S_i = M. \tag{III.17}$$

where S_i is a subnet. In practice, the subnet groupings in S can be obtained using a tool like ENV [44]. ENV (Effective Network View) uses a number of heuristics (e.g. bandwidth tests) to determine a logical representation of the network topology relative to a source machine. In our case, ENV groups M into subnets using the writer as the source machine; it also returns a subnet bandwidth to the source machine. For example, Figure III.6 shows the ENV representation of the network topology shown in Figure III.5. Using the logical network information provided by ENV, the following additional transfer constraint can then be introduced into our model:

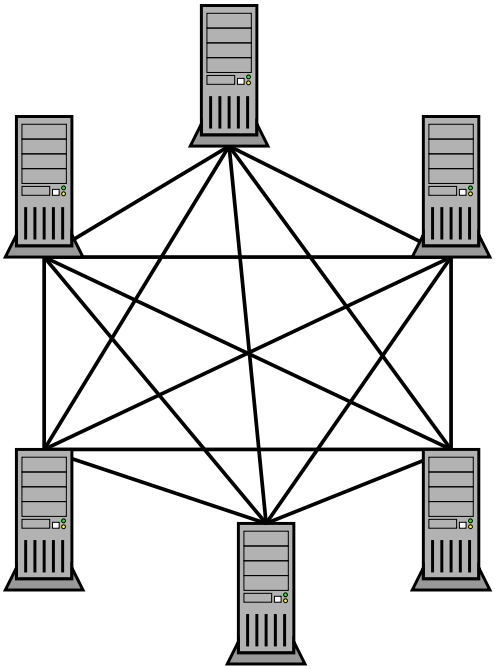


Figure III.4: Example of a fully connected network.

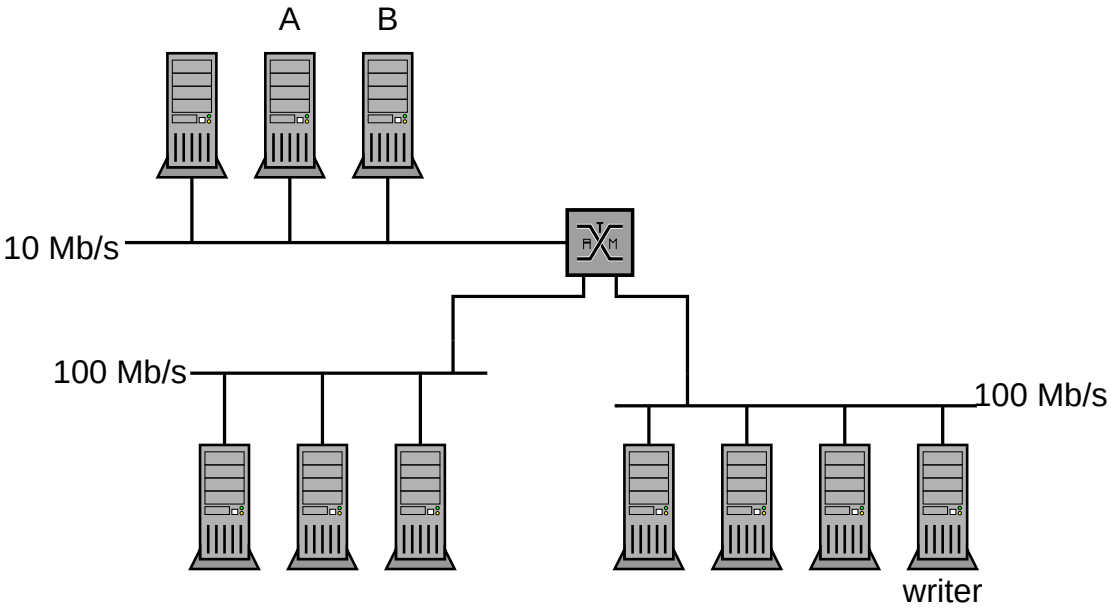


Figure III.5: Example of a LAN network topology.

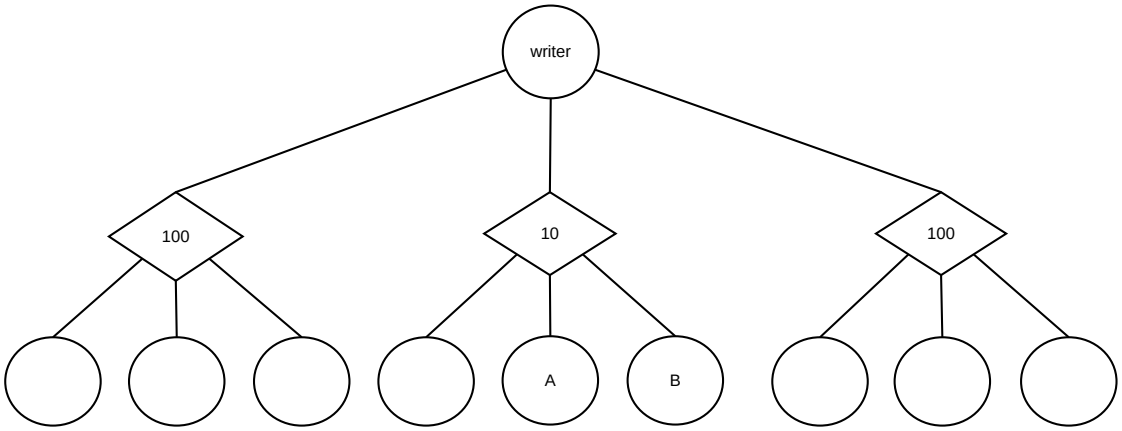


Figure III.6: ENV logical representation of the network topology shown in Figure III.5.

$$\forall S_i \in S \quad T_{comm}(S_i) \leq r \times a \tag{III.18}$$

where $T_{comm}(S_i)$ is the time in seconds for all compute resources in S_i to transfer $\sum_{m \in S_i} w_m$ slices to the writer. Therefore, we write

$$T_{comm}(S_i) \approx \frac{\left(\sum_{m \in S_i} w_m\right) \times \frac{x}{f} \times \frac{z}{f} \times sz}{B_{S_i}} \tag{III.19}$$

where B_{S_i} is the capacity (b/s) of the subnet link obtained from ENV. In other words, we want to allocate work to resources such that their cumulative transfers do not exceed the capacity of the subnet. Note that because we assume a heterogeneous network, Equations III.18 and III.19 complement Equations III.12 and III.16 rather than invalidating them.

In practice, there is no automated way to determine the bandwidth of the writer link using ENV unless one can ensure there is at least one other machine in M that is sharing the same link. Therefore, we do not include a constraint on the sum of the subnet transfers to the writer. However, if the bandwidth of the

writer link was available, it would be straightforward to add this constraint into the model.

Finally, we also do not introduce any transfer constraints into our model involving input data (i.e., projection data sent from the preprocessor to the ptomos). For the NCMIR scenarios, the input data is two orders of magnitude smaller than the output data (as noted in Section II.B) and its transfer time is amortized into the acquisition period. For other scenarios, this model could be extended in a straightforward way to include constraints on input data transfer.

III.C.4 Cost

In Section II.B.3, we defined cost in *service units* using the following equation:

$$su = charge\ factor \times number\ of\ CPUs \times wall\ clock\ time \quad \text{(III.20)}$$

Therefore, we add Equation III.20 to our system. Recall that in our model, a space-shared resource m is represented as a single resource (see Equation III.10). It is therefore possible that a space-shared resource will be allocated an amount of work that does not require the computational power of all u_m immediately available processors. In this case, we want to calculate how many processors are required to complete the computation for charging purposes. This is accomplished by calculating the time it would take to compute w_m on one processor of m and then dividing by the acquisition period, a.

$$n_m = \frac{tpp_m \times \frac{x}{f} \times \frac{z}{f} \times w_m}{a} \quad \text{(III.21)}$$

Since supercomputer centers do not charge for fractional pieces of CPU, we compute $\lceil n_m \rceil$. To express this in our equations, we add a slack variable, l_m, to the equation, where $0 \leq l_m < 1$, and constrain n_m to be an integer. Thus, n_m can be found using,

$$\frac{tpp_m \times \frac{x}{f} \times \frac{z}{f} \times w_m}{a} + l_m - n_m = 0 \tag{III.22}$$

Therefore, the following constraint can now be added to our model:

$$su = \sum_{m \in SSR} h_m \times n_m \times p \times a \tag{III.23}$$

where h_m is the charge factor, n_m is the number of CPUs used on m, and $p \times a$ is the wall clock time of execution. In other words, we sum together the service units using the charging policy of all resources in SSR. Note that,

$$\forall m \in SSR \quad n_m \leq u_m. \tag{III.24}$$

III.C.5 Putting it all together

The last set of constraints are the user constraints expressed in Equation III.1. We can now summarize our model in Figure III.7. Given this system of equalities and inequalities, determining W becomes an optimization problem. Recall from Section III.B, that we search for feasible triples by fixing two of the parameters and optimizing for the third. For convenience, we rewrite the three optimization problems from Section III.B here:

(i) fix f and r, minimize su;

(ii) fix r and su, minimize f; and

(iii) fix f and su, minimize r.

For both (i) and (iii), the system becomes linear upon substition of f. This is a clear advantage because there are numerous linear programming solvers freely available [29]. However, the system remains nonlinear for (ii). While nonlinear

programming solvers are also freely available [36], we opt to use a simpler technique. As a first approach, we exploit the discreteness and small range of f to reduce the nonlinear program to multiple linear programs using substitution. All linear systems are then solved using the **lp_solve** package [32] and the one with the optimal solution is chosen.

Ideally, an optimal solution would be found by formulating the linear program as an integer program.[1] An integer program is a linear program where all variables are constrained to be integers [2]. However, integer programs are harder to solve than linear programs [29]. Our experiments indicate that a mixed-integer approach, where w_m and l_m are expressed as continuous variables and all others as integer variables, is efficient. The drawback of this approach is that we have to round the values found for $w_m \in W$ since we cannot allocate fractional slices to ptomos. Therefore, the result is an *approximate* solution; we assess this in the following chapter.

III.D Summary

In this chapter, we defined a user-directed AppLeS. The AppLeS works by discovering feasible triples at run-time based on current resource availability and displays them as choices to the user. Once the user picks a triple, the AppLeS determines a work allocation. We then described how the AppLeS searches for triples and determines work allocation by characterizing scheduling/tuning as multiple constrained optimization problems. In the next chapter, we evaluate the performance of the AppLeS using simulations.

[1] Equation III.22 can be rewritten without l_m, the only continuous variable in our system.

$$\forall m \in M \qquad w_m \geq 0 \tag{1}$$

$$\sum_{m \in M} w_m = y \tag{2}$$

$$\forall m \in TSR \qquad \frac{tpp_m}{cpu_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m \leq a \tag{3}$$

$$\forall m \in SSR \qquad \frac{tpp_m}{u_m} \times \frac{x}{f} \times \frac{z}{f} \times w_m \leq a \tag{4}$$

$$\forall m \in M \qquad \frac{w_m \times (\frac{x}{f} \times \frac{z}{f} \times sz)}{B_m} \leq r \times a \tag{5}$$

$$\forall S_i \in S \qquad \frac{\sum_{m \in S_i} w_m \times \frac{x}{f} \times \frac{z}{f} \times sz}{B_{S_i}} \leq r \times a \tag{6}$$

$$\forall m \in SSR \quad \frac{tpp_m \times \frac{x}{f} \times \frac{z}{f} \times w_m}{a} + l_m - n_m = 0 \tag{7}$$

$$\forall m \in SSR \qquad n_m \leq u_m \tag{8}$$

$$su = \sum_{m \in SSR} h_m \times n_m \times p \times a \tag{9}$$

$$f_{min} \leq f \leq f_{max} \tag{10}$$

$$r_{min} \leq r \leq r_{max} \tag{11}$$

$$su_{min} \leq su \leq su_{max} \tag{12}$$

Figure III.7: The model of on-line parallel tomography.

Chapter IV

Experiments

IV.A Introduction

In this section we show three sets of results. In Section IV.B, we show that using dynamic load information improves scheduler performance. In the second set of results, described in Section IV.C, we demonstrate that tunability is an important characteristic for running on-line parallel tomography in a Computational Grid. Finally, we evaluate the scheduling latency of the AppLeS scheduler in Section IV.D.

IV.B Work Allocation

The goal of the first set of experiments was to investigate the impact of dynamic information on scheduler performance for on-line parallel tomography. For an experiment (45, 61, 1024, 1024, 300) as described in Sections I.A and I.B, we fix the application configuration (f, r, su) and compare the work allocation strategy of the AppLeS scheduler to schedulers which use no or partial dynamic information. In Table IV.1, we summarize the characteristics of the schedulers.

The first scheduler, wwa (weighted work allocation), corresponds to a very simple scheduling strategy that a user might employ to perform load balancing in

	infinite bandwidth	dynamic bandwidth
dedicated cpu	wwa	wwa+bw
dynamic cpu	wwa+cpu	AppLeS

Table IV.1: Summary of scheduler characteristics.

a heterogenous system. It performs work allocation based only on the relative processor benchmarks of the application in dedicated mode. This technique is considered simple because the only overhead is performing an application benchmark for each processor; this is a one-time only process and is something any user can perform.[1] In particular, this scheduling technique assumes no dynamic load information; i.e., it assumes dedicated processors and infinite bandwidth links.

The remaining schedulers build upon the wwa approach by assuming increasingly realistic characteristics about Grid resources. The scheduler wwa+cpu assumes that compute resources are shared among multiple users. It extends wwa by utilizing dynamic CPU load information. This corresponds to users who might run a system tool such as the UNIX command `uptime` on each machine to find out CPU availability before executing their application. The AppLeS scheduler, as described in Chapter III, assumes both compute and network resources are shared among multiple users. It builds upon wwa+cpu, by also utilizing dynamic bandwidth information. As explained in section III.C, dynamic CPU load and bandwidth information are obtained from the NWS. Note that some effort on the part of the user is required to set up and maintain the NWS sensors. The wwa+bw scheduler assumes only dynamic bandwidth information and no CPU load information.

IV.B.1 Performance Metric

Given the soft-real time requirement for on-line parallel tomography, we say that performance degrades when either the computation or communication

[1]The UNIX system call, `clock`, can be used to determine the approximate length of CPU time used by a process which can be used to approximate dedicated time.

soft deadlines, as described in Section III.C, are violated. Since the lateness of a computation deadline will effect the lateness of the communication deadline, we can summarize performance based on the refresh completion times. Therefore, we say that performance degrades when a refresh is late; that is, when a refresh's completion time is greater than the refresh period, $r \times a$. For each refresh, we measure the lateness relative to the lateness of the previous refresh. We call this *relative refresh lateness* (Δ_l) and use this as our performance metric for on-line parallel tomography. We now define Δ_l formally.

Let R = $\{1, ..., \frac{p}{r}\}$ be a set of refreshes for a single execution of on-line parallel tomography. Also, let $d(i)$ be the expected completion time (deadline) of a refresh $i \in R$ such that

$$d(i) - d(i-1) = r \times a. \tag{IV.1}$$

In other words, each refresh is expected to complete within the refresh period. Note that we assign $d(0) = 0$. Now let, $c(i)$ be the actual completion time of refresh i with respect to $d(0)$. If refresh $i-1$ is not late, then $\Delta_l(i)$ is simply the difference between the actual refresh completion time, $c(i)$, and the expected refresh completion time, $d(i)$:

$$\Delta_l(i) = c(i) - d(i). \tag{IV.2}$$

Substituting Equation IV.1 into Equation IV.2 gives

$$\Delta_l(i) = c(i) - d(i-1) - r \times a. \tag{IV.3}$$

Now, if refresh $i-1$ is late, then $c(i-1) > d(i-1)$ and we measure the lateness of refresh i relative to $c(i-1)$. Therefore,

$$\Delta_l(i) = c(i) - c(i-1) - r \times a. \tag{IV.4}$$

Combining Equations IV.3 and IV.4 gives the definition of $\Delta_l(i)$:

$$\Delta_l(i) = c(i) - max(d(i-1), c(i-1)) - r \times a. \quad \text{(IV.5)}$$

If $c(i)$ arrives early, then refresh i is not late and we define $\Delta_l = 0$. Therefore,

$$\Delta_l(i) = max[c(i) - max(d(i-1), c(i-1)) - r \times a, 0]. \quad \text{(IV.6)}$$

Note that if all refreshes arrive on time, each run will have $\frac{p}{r}$ refreshes. However, if any refreshes are late, it is likely that only a fraction of the refreshes will complete within data acquisition. Therefore, the total number of completed refreshes can also be a performance metric.

IV.B.2 Simulation

In order to compare scheduler performance, we must execute the application with each scheduler under the same environmental conditions. However, achieving reproducible environmental conditions is difficult in a dynamic environments [20]. One approach is to run experiments back-to-back in order to achieve *similar* environmental conditions [48, 46, 16]. Another approach is to use simulation [7].

Given the long makespan of on-line parallel tomography, achieving reproducible environmental conditions with back-to-back experiments is infeasible. Therefore, we conducted our experiments using simulation. This had the added benefit of allowing us to study the behavior of the schedulers in many different environments. We wrote a simulator using the Simgrid toolkit which provides a simulation API for studying scheduling algorithms in distributed systems [5]. Simgrid allows us to implement a discrete-event simulator and provides a notion of *tasks* (e.g. computation, data transfer) and *resources* (e.g. processors, network links). Tasks can have dependencies among them and are scheduled on resources.

Resources behaviors are modeled by service rates that can be modeled by traces from real resources (e.g. CPU availability, bandwidth of network link). Such traces are commonly available by existing resource monitoring tools such as the NWS. Furthermore, Simgrid makes it possible to create arbitrary resource interconnect topologies. The Simgrid approach has been verified in [5] and has been used to evaluate scheduling algorithms for parameter sweep applications [7, 8]. Similar trace-based resource simulation approaches have also been applied in projects such as Bricks [50].

In our simulator, we model four types of tasks based on profile information from the application:

acquire: acquire a projection from the microscope

scanline transfer: send a scanline from the preprocessor to a ptomo

backproject computation: backproject a scanline to a slice

slice transfer: send a slice from a ptomo to the writer

For a single simulation, there are p acquires. For each acquire, there are y scanline transfers and y backprojection computations. Given the value of r, there can also be y slice transfers following the backprojection computations. Resources are modeled as a Computational Grid containing multi-user workstations and space-shared supercomputers. The service rates workstations are modeled using NWS CPU availability traces taken from real machines. Similarly, the number of processors available on a supercomputer is taken from traces from a real supercomputer. Note that since we are modeling supercomputers as space-shared, processors of the supercomputer are modeled as having a constant service rate (i.e., no load). Simgrid allows us to create topologies in which workstations share the same network link to the preprocessor and writer; depending on the network topology, multiple workstations can also share the same network link to the preprocessor/writer. Similarly, dedicated processors on a supercomputer are modeled as sharing the

same network link to the preprocessor and writer. The service rates for network links are modeled using NWS bandwidth traces taken from real pairs of machines. Note, that in following Grid topology figures, we display the writer as the only I/O process even though we do simulate I/O from the preprocessor. That is, we display only relevant scheduler information (recall from Section III.C.3 that schedulers do not consider data transfers from the preprocessor to ptomo).

In Section IV.B.3, we show the results of simulations based on real traces from a cluster of workstations at NCMIR. These results indicate a relationship between the accuracy of predictions and scheduler performance which we study for a wider range of scenarios in Section IV.B.4.

IV.B.3 Case Study: NCMIR cluster

We first simulated experiments over a set of resources modeled after a real cluster of workstations at NCMIR. The machines are described in Table IV.2 and the network topology is shown in Figure IV.1.[2] The machine hamming was used as the writer machine because it had the highest bandwidth capacity. In Figure IV.2, we show the ENV representation of the topology relative to hamming. Note that due to the switched network and hamming's 1 Gb/s NIC, almost all machines appeared as if they had dedicated network links to hamming. The exceptions were golgi and crepitus which both have 100 Mb/s NICs. In this case, the ENV tool detected some network interference at the switch. We therefore modeled golgi and crepitus as sharing the same network link in our simulations.

To model the load on each resource, we collected CPU availability and bandwidth traces using the NWS on March 8th, 2001 from 8:00 A.M. to 4:00 P.M. PST. This corresponds to a workday during which users at NCMIR would run on-line parallel tomography. The sample period for both CPU availability and bandwidth were set to the NWS defaults, 10 and 120 seconds respectively. The

[2]There are other machines not included in our simulation that are connected to both switches; two other machines are connected to the Cisco 2916 XL switch and 11 other machines are connected to the Cisco 6509 switch.

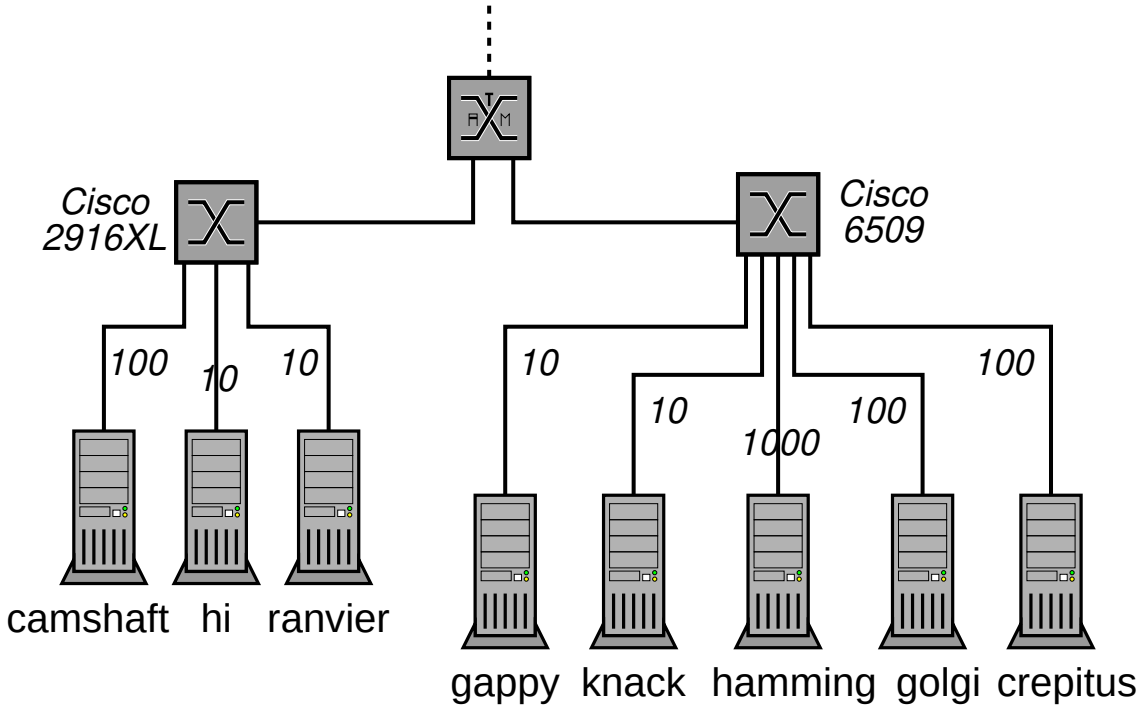


Figure IV.1: Network topology of a cluster of machines at NCMIR.

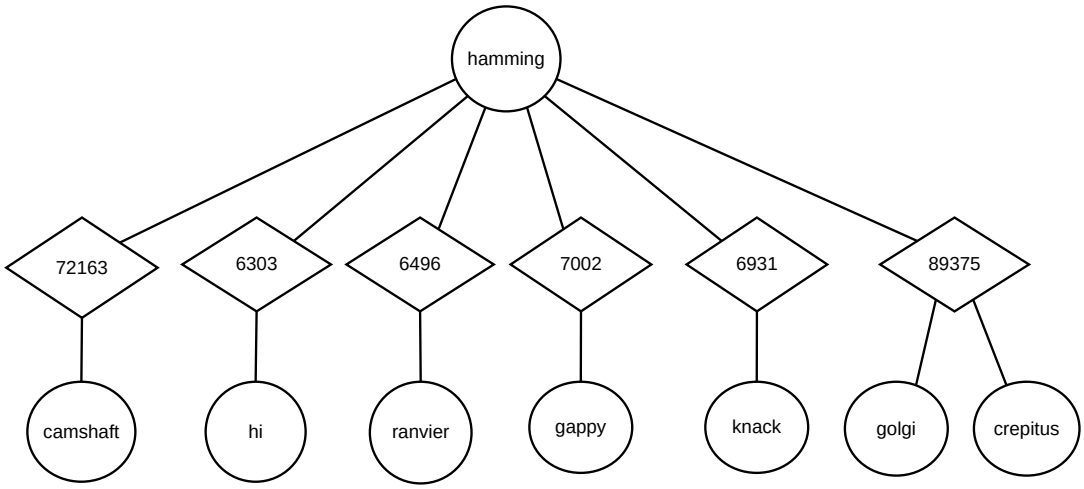


Figure IV.2: ENV representation of NCMIR topology. The numbers in the diamonds are the subnet bandwidths (Kb/s) found by ENV.

Name	Manufacturer	Model	Processor	Speed	Memory
camshaft	Sun	Ultra-60	UltraSPARC-II	295 MHz	384 MB
gappy	SGI	Indigo2	MIPS R10000	175 MHz	384 MB
golgi	SGI	Octane	MIPS R10000	250 MHz	2 GB
knack	SGI	Indigo2	MIPS R4400	200 MHz	128 MB
crepitus	SGI	Octane	MIPS R10000	250 MHz	2 GB
ranvier	SGI	Indigo2	MIPS R4400	200 MHz	128 MB
hi	SGI	Indigo2	MIPS R10000	195 MHz	512 MB
hamming	Sun	Ultra-80	UltraSPARC-II	450 MHz (2)	4 GB

Table IV.2: NCMIR machine descriptions.

	mean	std	cv	min	max
camshaft	43.432	3.988	0.092	10.758	51.925
gappy	7.122	2.309	0.324	2.764	9.126
knack	7.119	2.371	0.333	2.149	9.007
golgi/crepitus	77.218	8.845	0.115	5.113	80.179
ranvier	6.911	2.220	0.321	2.611	8.899
hi	8.921	0.376	0.042	3.618	9.072

Table IV.3: Summary statistics for the bandwidth traces (Mb/s) displayed in Figure IV.3.

traces are displayed in Figures IV.3 and IV.4. Summary statistics for the traces are displayed in Tables IV.3 and IV.4. For each trace, the table shows the mean (*mean*), the standard deviation (*std*), the coefficient of variance (*cv*), the minimum (*min*), and the maximum (*max*) trace values. We conducted two sets of simulations at 10 minute intervals throughout the simulated 8 hour period. In the first set of simulations described in Section IV.B.3.1, we simulate runs where the schedulers have perfect load predictions; this is accomplished by running partially trace-driven simulations. In the second set of simulations described in Section IV.B.3.2, we allow the load on resources to vary according to the traces.

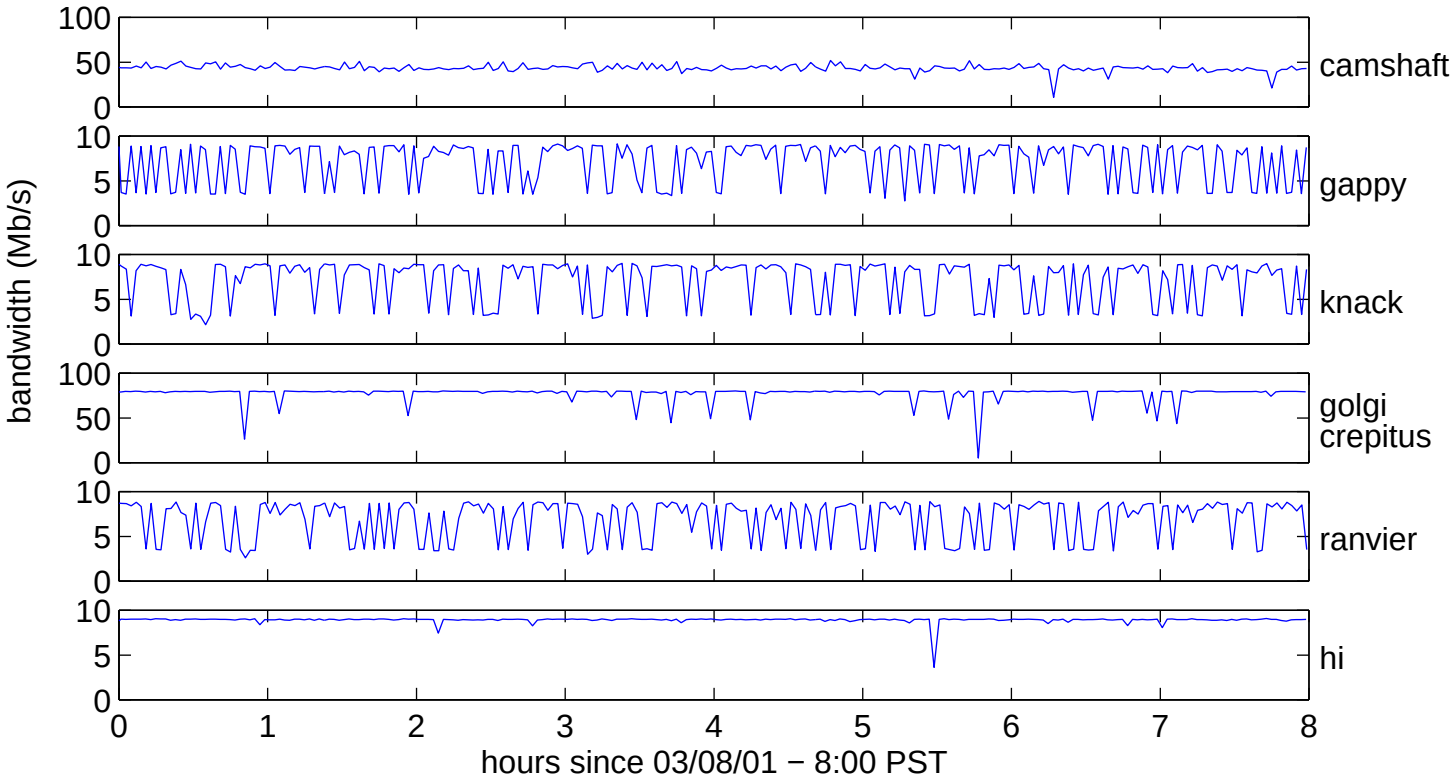


Figure IV.3: NWS Bandwidth traces taken from NCMIR machines on March 8th, 2001 from 8:00 A.M. to 4:00 P.M. PST.

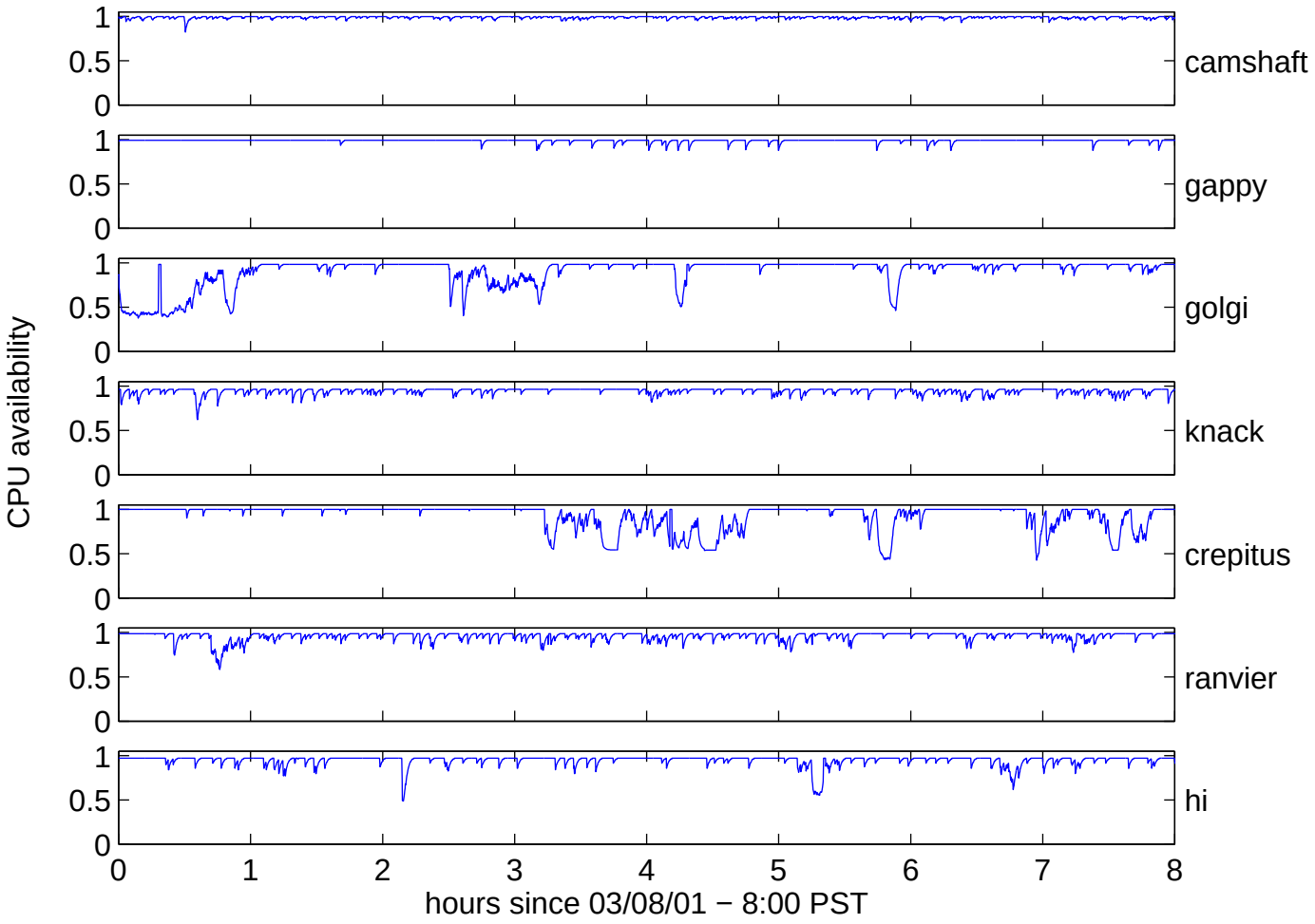


Figure IV.4: NWS CPU availability traces taken from NCMIR machines on March 8th, 2001 from 8:00 A.M. to 4:00 P.M. PST.

	mean	std	cv	min	max
camshaft	0.988	0.013	0.013	0.824	0.997
gappy	0.988	0.017	0.017	0.878	0.993
golgi	0.902	0.157	0.174	0.376	0.984
knack	0.944	0.032	0.034	0.622	0.964
crepitus	0.925	0.136	0.147	0.427	1.000
ranvier	0.958	0.047	0.049	0.582	0.987
hi	0.946	0.059	0.062	0.487	0.972

Table IV.4: Summary statistics for the CPU availability traces displayed in Figure IV.4.

IV.B.3.1 Partially Trace-driven Simulations

In this set of experiments, we simulated runs where the schedulers had access to perfect load predictions. This represents the optimal running environment for the schedulers since the scheduling decision made at the beginning of execution was good throughout execution. At the start of each simulation, we used the trace to determine a constant resource load for the duration of the simulation. Therefore, we say the simulations are *partially* trace-driven. In Figure IV.5, we show the results of the simulations by plotting the mean relative refresh lateness for each scheduler over the eight hour simulation period. From this figure, it is clear that the AppLeS scheduler outperforms all the other schedulers. It is followed by the wwa+bw scheduler which outperforms both the wwa and wwa+cpu schedulers indicating that communication is the dominant factor in application performance. The almost identical performance of the wwa and wwa+cpu schedulers further illustrates this in that the performance degradation due to bandwidth misprediction experienced by the wwa scheduler dominates the CPU availability misprediction. Note that for these resources, assuming 100% CPU availability does not result in significantly high errors due to the high fraction of CPU availability on the NCMIR machines (see Table IV.4).

In Figure IV.6, we show the distribution of Δ_l for all refreshes. For each scheduler, we plot the cumulative distribution function of Δ_l. A point (x, y) on the

scheduler	wwa	wwa+cpu	wwa+bw	AppLeS
count	337	341	1383	1449
% late	0.8220	0.7419	0.2061	0.0524
mean	290.8878	285.4264	3.8361	0.0004
std	739.2997	735.2143	13.9819	0.0121
min	0.0000	0.0000	0.0000	0.0000
max	2610.0000	2610.0000	96.8300	0.4580
median	57.8400	53.7220	0.0000	0.0000

Table IV.5: Summary statistics for NCMIR simulations with perfect load predictions.

graph represents that y percent of the refreshes were less than x seconds late. Here again we see the almost identical performance of the wwa and wwa+cpu schedulers, although wwa+cpu has a higher fraction of small Δ_l. For the wwa+bw scheduler, we see that most refreshes are under 10 seconds late with the rest under about 100 seconds late. Finally, the AppLeS scheduler shows the best performance with all Δ_l under one second late.

Summary statistics for each scheduler are displayed in Table IV.5. For each scheduler, the table shows the number of completed refreshes over all runs (*count*), the fraction of refreshes that were late (*% late*), the mean Δ_l (*mean*), the standard deviation of Δ_l (*std*), the minimum Δ_l (*min*), the maximum Δ_l (*max*), and the median Δ_l (*median*). The table shows more precisely that only 5% of the refreshes arrived late for the AppLeS scheduler (in fact, all were under a half a second late). Therefore, the approximate solution approach described in the previous chapter only marginally affected performance. So, we conclude that with perfect load predictions, the AppLeS scheduler had *near perfect* real-time performance. We now consider simulations where load predictions may be imperfect.

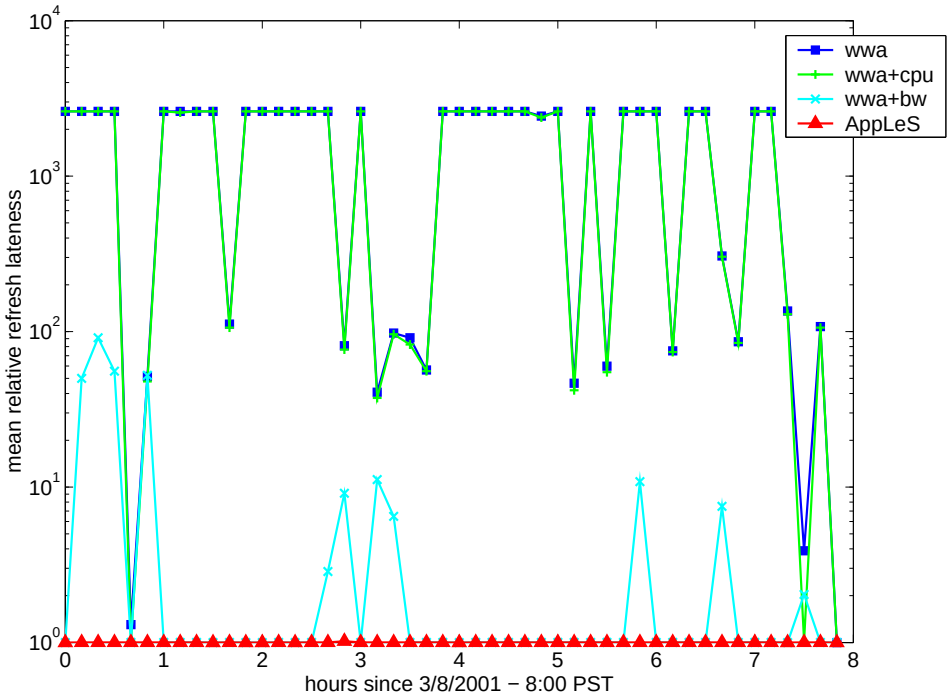


Figure IV.5: Simulation results with perfect load predictions. The mean relative refresh lateness for each scheduler is plotted over an 8 hour simulation period.

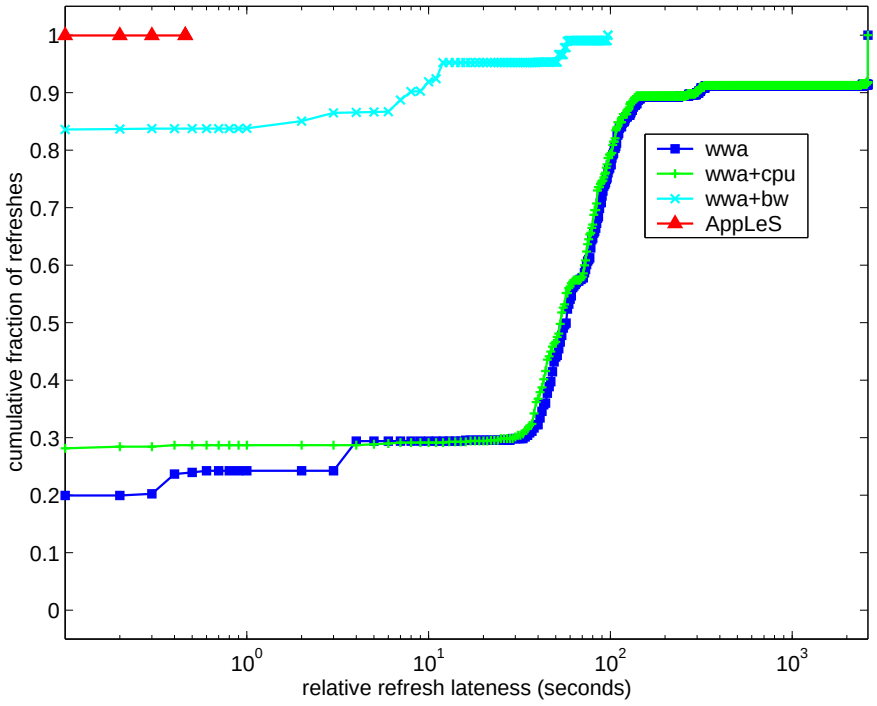


Figure IV.6: Simulation results with perfect load predictions. The cumulative distribution functions of Δ_l for each scheduler.

scheduler	wwa	wwa+cpu	wwa+bw	AppLeS
count	331	338	1160	1156
% late	0.87	0.88	0.58	0.58
mean	287.47	277.91	27.16	27.67
std	694.87	683.39	48.19	48.89
min	0.00	0.00	0.00	0.00
max	2610.00	2610.00	466.93	466.93
median	78.36	74.33	2.94	2.95

Table IV.6: Summary statistics for NCMIR simulations with imperfect load predictions.

IV.B.3.2 Completely Trace-driven Simulations

In this set of experiments, we used traces to determine resource load variation throughout simulation. Therefore, these simulations are *completely* trace-driven. Consequently, the initial load predictions may be imperfect throughout the simulated period. The results of the simulations are displayed in a mean relative refresh lateness plot shown in Figure IV.7 and a cumulative distribution function plot shown in Figure IV.8. Here again, we see that the wwa and wwa+cpu schedulers have nearly identical performance. Furthermore, the wwa+bw and AppLeS scheduler also have nearly identical performance. Comparing this to the previous set of simulations, we see how imperfect predictions impact the performance of the AppLeS scheduler.

Summary statistics for the simulations are shown in Table IV.6. From these numbers, we see that the wwa+cpu scheduler outperforms the wwa scheduler indicating no and/or negligible CPU availability mispredictions. However, using CPU availability predictions does not seem to benefit the AppLeS scheduler in the same way.

In Figure IV.7, the mean relative refresh lateness is lowest for the wwa+bw scheduler five times (at .167, .333, .833, 2.833, and 3.333 hours). Upon further investigation, we found that the AppLeS' performance drop was not a result of CPU availability mispredictions (the wwa+cpu scheduler outperforms the wwa sched-

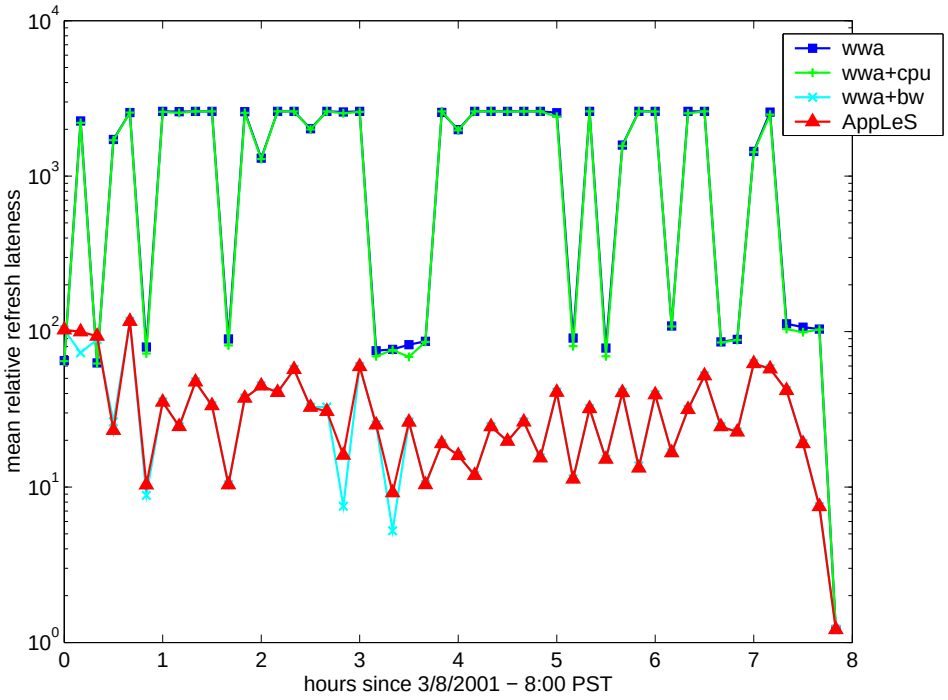


Figure IV.7: Simulation results with imperfect load predictions. The mean relative refresh lateness for each scheduler is plotted over an 8 hour simulation period.

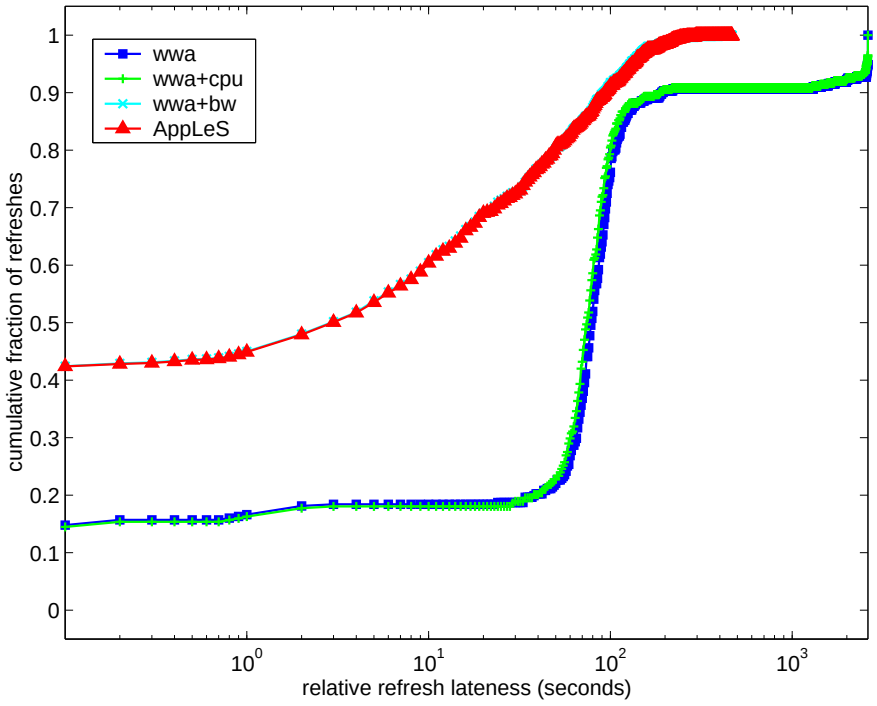


Figure IV.8: Simulation results with imperfect load predictions. The cumulative distribution functions of Δ_l for each scheduler.

uler). Rather the performance drop was a result of bandwidth mispredictions. In all simulations, the wwa+bw and AppLeS schedulers allocated work to machines camshaft, gappy, and golgi. In the cases where the AppLeS scheduler detected a drop in CPU availability on the machine golgi, work was also allocated to knack and sometimes crepitus. From Figure IV.3, we can see that network bandwidth to knack is much more variable than to golgi; as a result the AppLeS scheduler mispredicted the bandwidth availability resulting in a worse work allocation than the wwa+bw's work allocation.

However, we note that the mispredictions made by the AppLeS scheduler resulted in marginal degradation for the most part compared to the wwa+bw scheduler; in the five cases listed above, the difference in the average mean relative refresh lateness was 26.7, 4.4, 1.4, 8.5, and 4.0 seconds. However, we also note that the the mean relative refresh lateness for the AppLeS scheduler is 27.6710 seconds higher than in the perfectly predicted simulations. This is most likely the result of mispredicting bandwidth availability to gappy (one of the more variable traces). Chapter VI discusses a couple of approaches to address this problem. In the next section, we further investigate the impact of bandwidth mispredictions on scheduler performance.

IV.B.4 Synthesized Grid Experiments

In the previous section, we found that bandwidth predictions had a higher impact on scheduler performance than CPU availability predictions. Therefore, in this section, we provide a more general discussion of the impact of bandwidth predictability on scheduler performance. Rather than studying additional snapshots of real Grids as done in the previous section, we *synthesize* Grids using real CPU availability and bandwidth traces. This allows us to study a wider range of network behaviors. In Section IV.B.4.1, we discuss how we categorize and construct a Grid using a bandwidth predictability metric. Section IV.B.4.2 discusses relative scheduler performance and Sections IV.B.4.3 and IV.B.4.4 discuss the results of

the simulations in terms of bandwidth predictability.

IV.B.4.1 Grid Construction

In order to study the impact of bandwidth predictability on scheduler performance, we fixed the topology of the Grids in order to have comparable results. The topology we used is illustrated in Figure IV.9; it contains three clusters of workstations composed of 8, 8, and 16 hosts respectively. All hosts in the cluster shared one network link to the writer machine. We then varied the traces used for each Grid in order to exhibit different network behaviors. For all simulations, the CPU availability of each host and the bandwidth of each network link were completely trace-based.

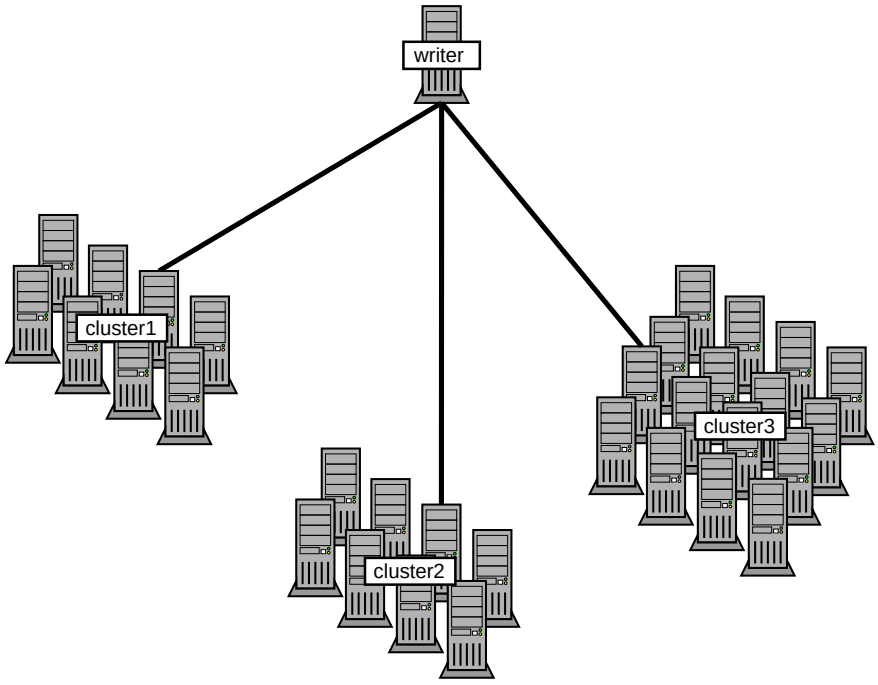


Figure IV.9: Grid topology for work allocation simulations.

The traces we used were collected from various research sites across the United States and Europe using the NWS. For bandwidth, we collected 429 traces from 66 machines spread across 12 sites using the NWS default sample period of

2 minutes. These traces were collected during the period February 10 - 27, 2001. We then processed these traces for gaps (i.e., missed measurements) and divided them into 546 continuous trace segments such that the elapsed time between two successive measurements was no more than 6 minutes. CPU availability traces were taken from 100 machines spread over 17 sites using the NWS default sample period of 10 seconds. These traces were collected during the period August 31 to October 25, 1999 and were processed into 1021 trace segments such that the elapsed time between successive measurements was no more than 5 minutes.

In order to classify the traces we collected, we needed a method to characterize the predictability of a trace. Determining the predictability of a trace for on-line parallel tomography is difficult due to its long makespan and the lack of long-range forecasters. Given that our scheduler uses short-term forecasts provided by the NWS, we estimated that the predictability of a trace would be correlated to the variability of a trace for applications with long makespans. Therefore, we approximate predictability using the coefficient of variance.

For bandwidth, we plotted the histogram shown in Figure IV.10. The histogram shows three *clusters* which we used to divide the traces into three categories: [0, 0.20), [0.20, 0.45), and [0.45, 0.70). We labeled the categories *high*, *medium*, and *low* predictability respectively. We then picked ten trace segments from each category where each trace ranged from two to eight days in length. To substantiate the use of the coefficient of variance, we calculated the *average prediction error* for each of the ten traces. To determine the average prediction error, we used the NWS forecaster library to calculate the predicted value, v_p, for each actual trace value, v_a. The average prediction error, $\bar{e}_p$ is then calculated for n measurements using

$$\bar{e}_p = \frac{\sum_{i=1}^{n} |v_p(i) - v_a(i)|}{n}. \tag{IV.7}$$

In Figure IV.11, we plot the average prediction error versus the coefficient

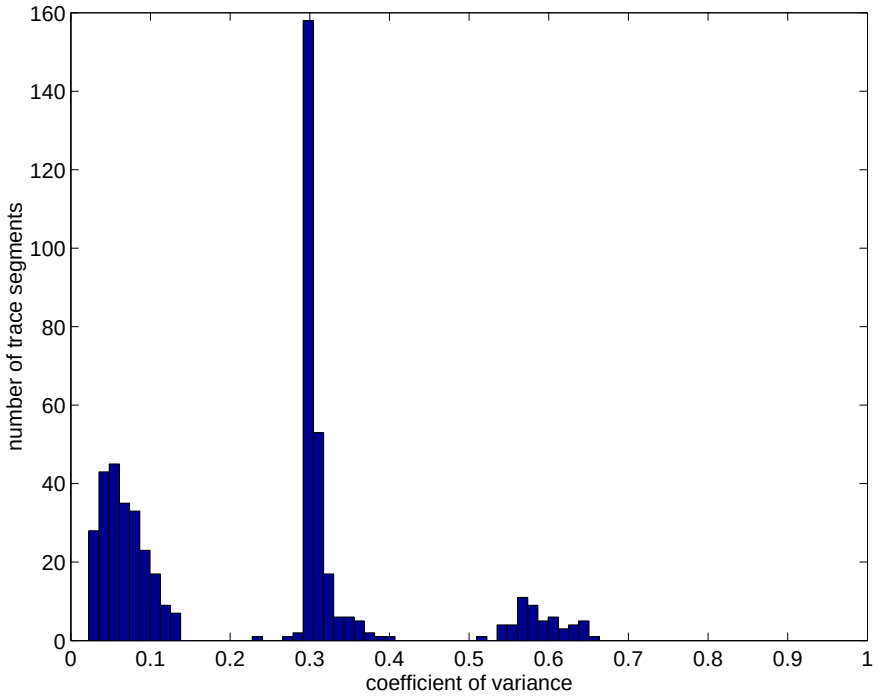


Figure IV.10: Coefficient of variance histogram for bandwidth traces.

of variance for each trace. The highest cluster of green triangle points correspond to the traces categorized as *low* predictability, the middle cluster of red diamond points correspond to the traces of *medium* predictability, and the lowest cluster of blue square points correspond to the traces of *high* predictability. From this graph, we see that there is a high correlation between the coefficient of variance and the average prediction error. However, we emphasize that this technique is a coarse measurement of predictability; we will discuss cases where this technique did not sufficiently capture the predictability of a trace in Section IV.B.4.3.

For CPU availability, we applied the same technique as done for the bandwidth traces. The coefficient of variance histogram is plotted in Figure IV.12 which we used to divide the traces into three categories: [0, 0.2), [.2 .4), and [.4, .5]. We then picked fifteen traces from each category. However, the CPU availability traces did not exhibit the same correlation to average prediction error as the bandwidth traces (see Figure IV.13). As a result of this and because we are mostly inter-

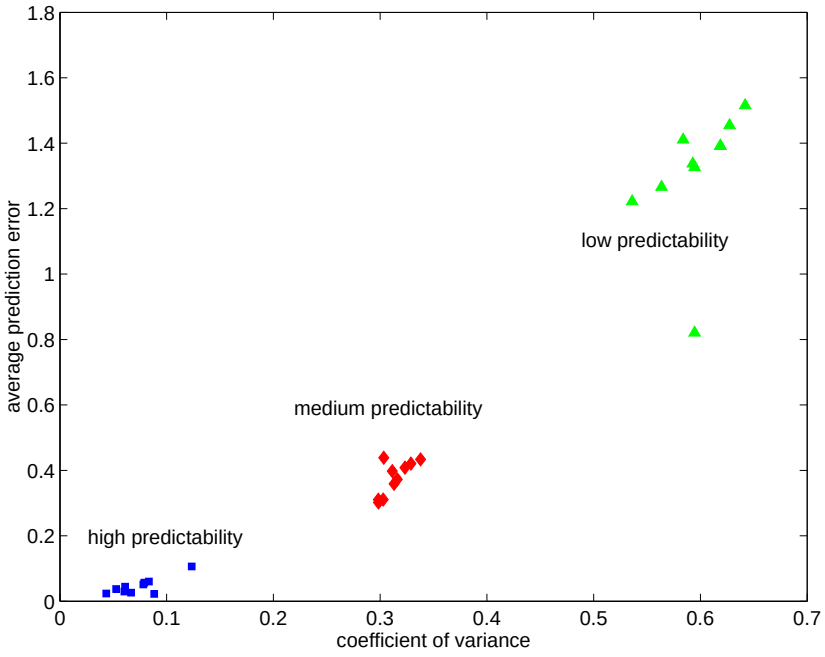


Figure IV.11: Correlation between coefficient of variance and average prediction error for bandwidth traces.

ested in bandwidth predictability (see Section IV.B.3.2), we therefore characterize a Grid by the types of traces used for bandwidth. In other words, a Grid can be described using a triple, (p_1, p_2, p_3), where $p_1, p_2, p_3 \in \{low, medium, high\}$; p_1 is the type of bandwidth trace used for the network link between cluster 1 and the writer, p_2 is the type of trace used for cluster 2, and p_3 is the type of trace used for cluster 3. For convenience, we abbreviate $low, medium$, and $high$ as L, M, and H respectively and write a tuple as $p_1p_2p_3$ (e.g. LHM).

Given the triple $p_1p_2p_3$, there are 27 different types of Grids. We randomly generated a total of 2510 different Grids (Table IV.7 shows the number of Grids that were generated for each Grid type). Since there are four schedulers, this resulted in a total of 10,040 simulations. The results of these simulations follow in the next three subsections.

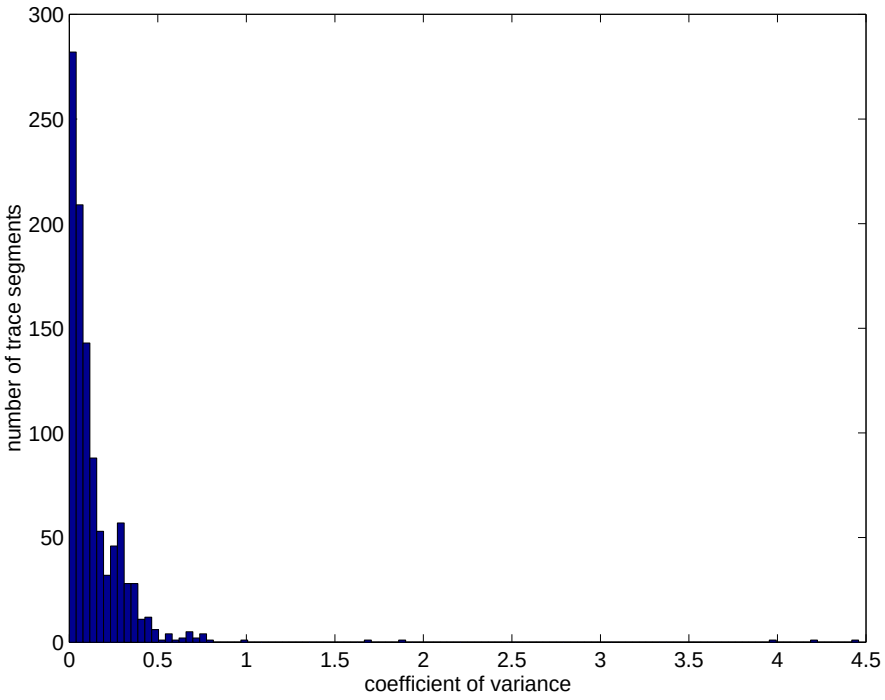


Figure IV.12: Coefficient of variance histogram for CPU availability traces.

Grid type	# of Grids
LLL	100
LLM	130
LLH	140
LML	70
LMM	100
LMH	120
LHL	90
LHM	80
LHH	110
MLL	70
MLM	120
MLH	90
MML	70
MMM	100
MMH	90
MHL	50
MHM	100
MHH	100
HLL	60
HLM	120
HLH	80
HML	120
HMM	70
HMH	40
HHL	130
HHM	90
HHH	70

Table IV.7: Number of Grids generated for each Grid type $p_1p_2p_3$.

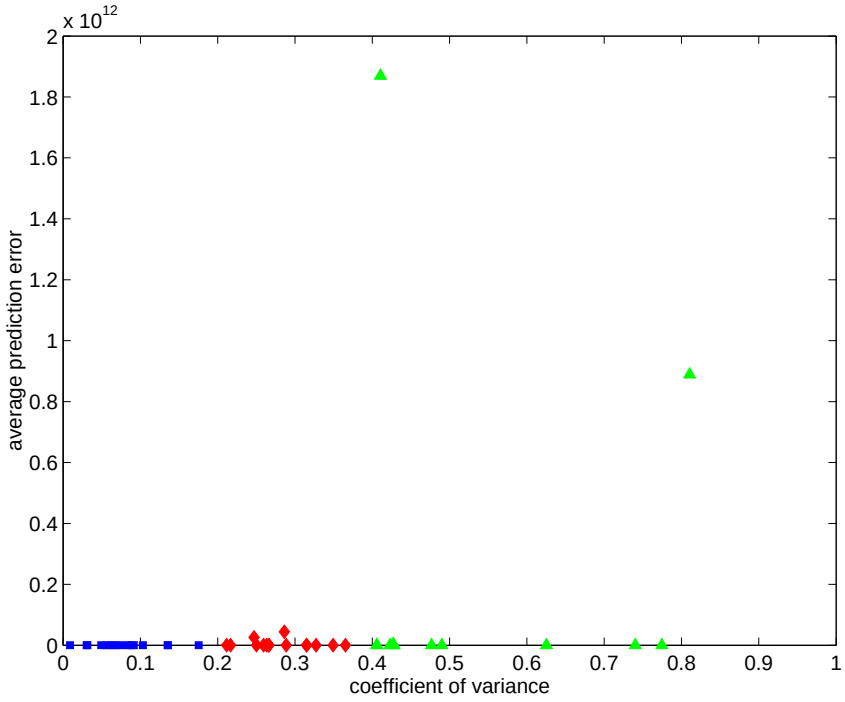


Figure IV.13: No correlation between coefficient of variance and average prediction error for CPU availability traces.

	1st	2nd	3rd	4th
wwa	182	97	1770	461
wwa+cpu	452	139	1164	755
wwa+bw	1105	900	412	93
AppLeS	2077	376	50	7
total	3816	1512	3396	1316

Table IV.8: Scheduler ranking based on cumulative Δ_l for synthetic Grid simulations. The table displays the number of times a scheduler ranked first, second, third, and fourth place.

IV.B.4.2 Scheduler Comparisons

To compare the simulation results for the schedulers on a run-to-run basis, we plotted the number of times each scheduler ranked first, second, third, and fourth place in a stacked bar graph in Figure IV.14; the ranking is based on cumulative relative refresh lateness ($\sum \Delta_l$) for each run. Values for the graph are displayed in Table IV.8. Ranking for this graph was performed as follows:

1. For a single run, scheduler i received a rank k if $k-1$ schedulers beat it.

2. For a single run, if more than one scheduler had the the same cumulative relative refresh lateness, they received the same rank.

Here it is clear that the AppLeS scheduler performed better than all other schedulers. The wwa+bw scheduler followed as second. However, the relative performance between the wwa and wwa+cpu scheduler is unclear; wwa+cpu is first more frequently than wwa but is also last more frequently than wwa. Therefore, for each scheduler, we calculated the average deviation from best scheduler in Table IV.9. Here, we see that wwa+cpu beat wwa by 46.98 seconds; therefore, when wwa+cpu is in last place (from mispredictions), it is not far from third place. Furthermore, we see that the AppLeS scheduler beat wwa+bw by 126.03 seconds.

Figure IV.15 shows the cumulative distribution functions of Δ_l over all 10,040 simulations; the results are grouped by scheduler. This shows that the Ap-

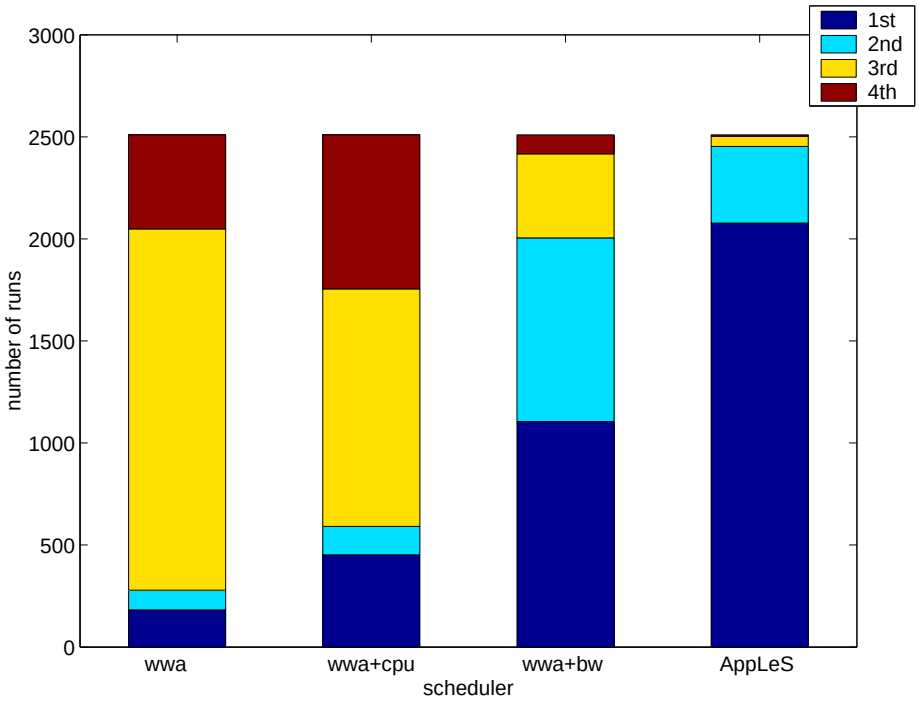


Figure IV.14: Scheduler ranking based on cumulative Δ_l.

wwa	wwa+cpu	wwa+bw	AppLeS
705.89	658.91	127.10	1.07

Table IV.9: Average deviation from best scheduler based on cumulative Δ_l for synthetic Grid simulations.

scheduler	wwa	wwa+cpu	wwa+bw	AppLeS
count	29721	35024	64621	81583
% late	0.80	0.68	0.71	0.59
mean	145.11	116.93	36.76	18.54
std	400.00	348.38	106.55	50.75
min	0.00	0.00	0.00	0.00
max	2745.00	2745.00	2655.00	2655.00
median	16.27	1.075	7.88	0.37

Table IV.10: Summary statistics for synthetic Grid simulations.

pLeS scheduler has the highest fraction of small Δ_l and the lowest fraction of large Δ_l; conversely the wwa scheduler has the smallest fraction of small Δ_l and the highest fraction of large Δ_l. We also see that the wwa+cpu and wwa+bw's cumulative distribution functions cross over each other; this shows that the wwa+cpu scheduler has a higher fraction of low Δ_l compared to wwa+bw, but also has a higher fraction of high Δ_l. In Table IV.10, we display summary statistics for the simulations. These statistics also show that the AppLeS scheduler outperforms all other schedulers. Furthermore, they show that the wwa+cpu scheduler outperforms wwa. However, the relative performance between the wwa+cpu and wwa+bw schedulers is not as clear. The wwa+cpu scheduler has a smaller fraction of late refreshes and a smaller median than wwa+bw; however, the wwa+bw scheduler has completed significantly more refreshes and exhibits a lower mean and standard deviation. Therefore, we say that both CPU availability and bandwidth predictions can improve scheduler performance. However, we conclude that bandwidth predictions are more important because when refreshes are late for wwa+bw, they are late by a smaller amount than wwa+cpu's. This is further supported in Figure IV.14, where the AppLeS and wwa+bw schedulers dominate first and second place.

Note that the benefit of CPU availability predictions is more apparent in these simulations because of a more diverse set of CPU availability traces (recall that for the NCMIR simulations described in Section IV.B.3 the mean CPU availability for each machine was at least .90). See Figure IV.16 for a histogram of the

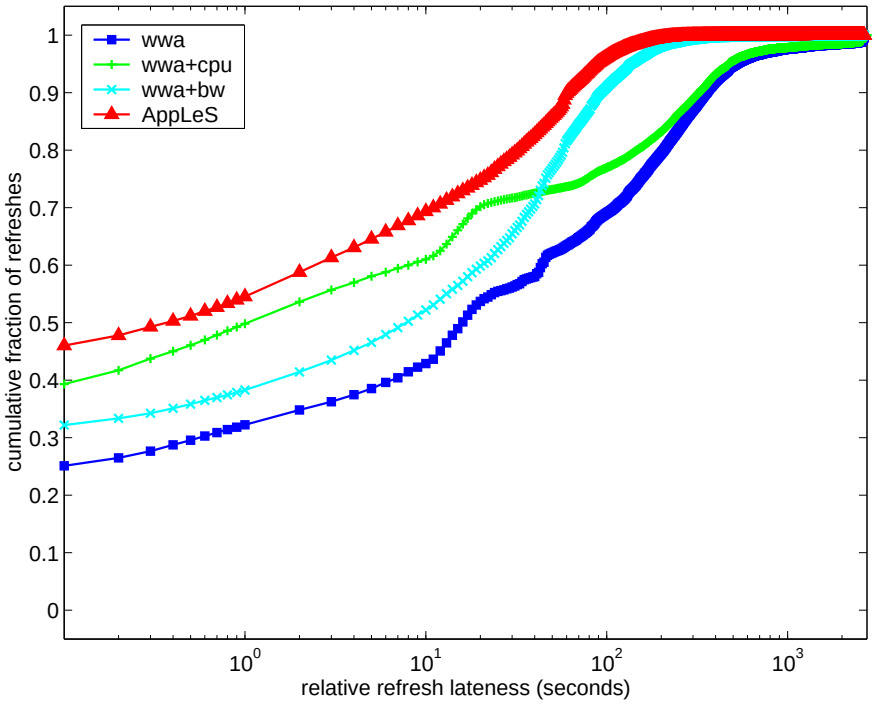


Figure IV.15: Synthetic Grid simulation results: the cumulative distribution functions of Δ_l for each scheduler.

mean CPU availability of the traces.

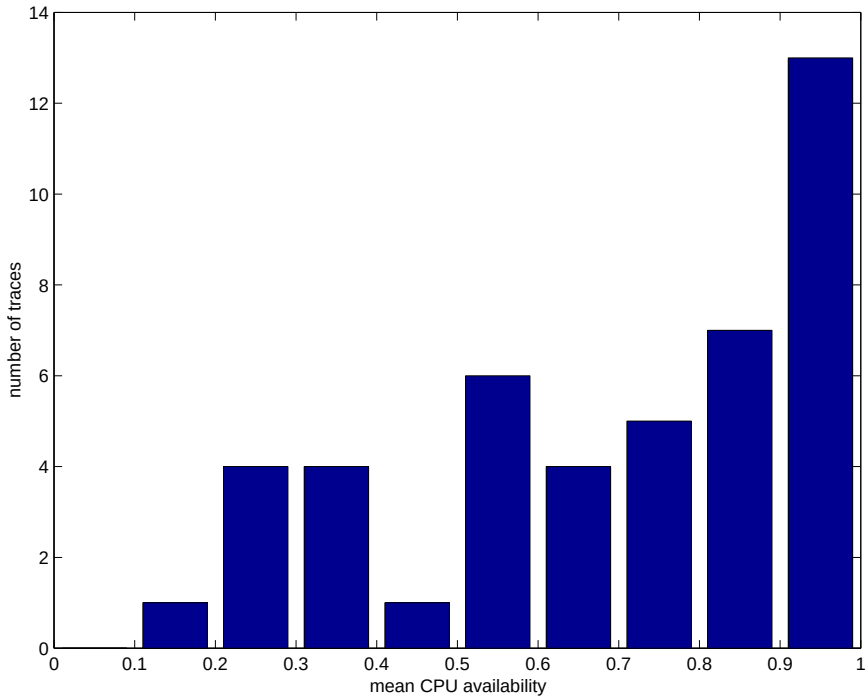


Figure IV.16: Mean of CPU availability for traces used in synthetic runs.

Given the importance of bandwidth predictions identified above, we now study how the quality of bandwidth predictions affect scheduler performance.

IV.B.4.3 Partial Orders

To study how quality of bandwidth predictions effect scheduler performance, we look at how scheduler performance degrades as the quality of bandwidth predictions degrades. Since there is no clear way to summarize bandwidth predictability for a Grid, we characterize a Grid's bandwidth predictability using the triple $p_1p_2p_3$. Then we compare simulation results that form a partial order. For example, we say that

$$(HHL, HML, HLL, LLL)$$

forms a decreasing partial order because for each successive triple, there is at least one trace with lower predictability than its predecessor and no traces with a higher predictability (note that $H > M > L$). Therefore, we say all triples in the partial order are *comparable* and compare the simulation results only between comparable triples. We make no assumptions about triples containing lower and higher predictability traces than the other. For example, we say the triples HLM and HML are *not comparable* because HLM has a trace with a lower predictability than HML's ($L < M$), and a trace with a higher predictability than HML's ($M > L$). We define the partial order more formally below.

As described in Section IV.B.4.1, we denote the predictability of a trace to be p, where $p \in P = \{L, M, H\}$. A total ordering on the set P is

$$(L, M, H). \tag{IV.8}$$

For triples $a, b \in P_1 \times P_2 \times P_3$ where $a = (a_1, a_2, a_3)$ and $b = (b_1, b_2, b_3)$, we define a relation R, a is more predictable than b, as

$$a \geq b, \quad \text{if} \quad a_i \geq b_i, i = 1, 2, 3. \tag{IV.9}$$

The relation R is reflexive, symmetric, and transitive, and therefore is a partial order [11]. For example, $HHM \geq HHL$. However, $HLM \ngeq LHM$ and $LHM \ngeq HLM$. We say HLM and LHM are not comparable.

We now look at the simulation results using decreasing partial orders. Figure IV.17 shows the results for 7 partially ordered triples:

$$P_1 = (HHH,\ HHM,\ HMM,\ HLM,\ MLM,\ LLM,\ LLL). \tag{IV.10}$$

Each triple in the partial order, P_1, is represented by a group of 4 boxplots. The boxplots are ordered from left to right and represent the wwa, wwa+cpu, wwa+bw,

and AppLeS schedulers respectively. Each boxplot is computed from 70 simulations. The square on the boxplot represents the mean Δ_l, the lower bar represents the minimum Δ_l, and the upper bar represents the maximum Δ_l. From this graph, we see that the mean Δ_l for the AppLeS scheduler is quite good at 3 seconds for the HHH Grids. Then the Δ_l increases by 14 seconds for the HHM Grids and continues to increase until it levels off at about 35 seconds. This shows that performance of the AppLeS and other schedulers degrades as Grid predictability degrades.

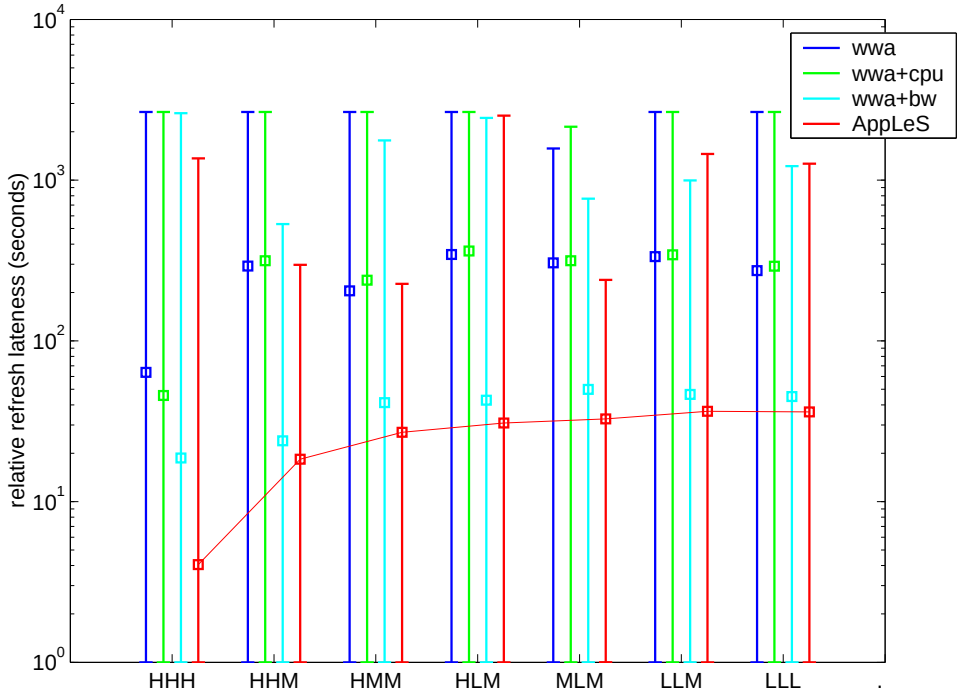


Figure IV.17: Simulation results grouped by partial order P_1. Each boxplot summarizes 70 simulations.

However, not all decreasing partial orders experience an increase in the mean Δ_l. Figure IV.18 shows a partial order, P_2:

$$P_2 = (HMH,\ MMH,\ LMH,\ LLH). \tag{IV.11}$$

Each boxplot in Figure IV.18 is computed from 40 simulations. Here the mean Δ_l of MML is 7 seconds lower than LML and 4 seconds lower than HML. In this case, the performance of the AppLeS scheduler was not monotonically decreasing. We attribute this is to our coarse predictability classification technique as noted in Section IV.B.4.1. For example, a low predictable trace (using the classification scheme in Section IV.B.4.1) does not always imply bad predictions. To illustrate, consider the trace segment characterized as having low predictability displayed in Figure IV.19. The upper plot shows a bandwidth trace taken from torc8.cs.utk.edu to torc4.cs.utk.edu during the morning of February 12, 2001. The lower plot is a prediction of the upper trace that was generated using the NWS forecaster library. Now consider the average bandwidth over a period of 45 minutes (the minimum time it would take to acquire a data set from NCMIR's electron microscope). Depending on when the scheduler queries for a bandwidth forecast, the scheduler might get point O, an overestimate of the average bandwidth; U, an underestimate of the average bandwidth; or G, a good estimate of the average bandwidth.

In summary, we demonstrated that under the predictability classification outlined in Section IV.B.4.1, the performance of the AppLeS scheduler degrades as the quality of bandwidth predictions degrades. We also noted a case where the performance of the AppLeS scheduler did not monotonically degrade thereby illustrating the coarseness of our predictability measurement. However, as we discuss in the next section, if we consider the results of all simulations, we see that our predictability measurement does demonstrate that the performance of the AppLeS scheduler does degrade as the quality of bandwidth predictions degrades.

IV.B.4.4 Scoring

In this section, we consider the performance of the AppLeS scheduler and quality of bandwidth predictions over *all* simulations. We use an arbitrary scoring technique to coarsely summarize the predictability of a Grid. Then, we assign a predictability score to each type of Grid and group results with the same score.

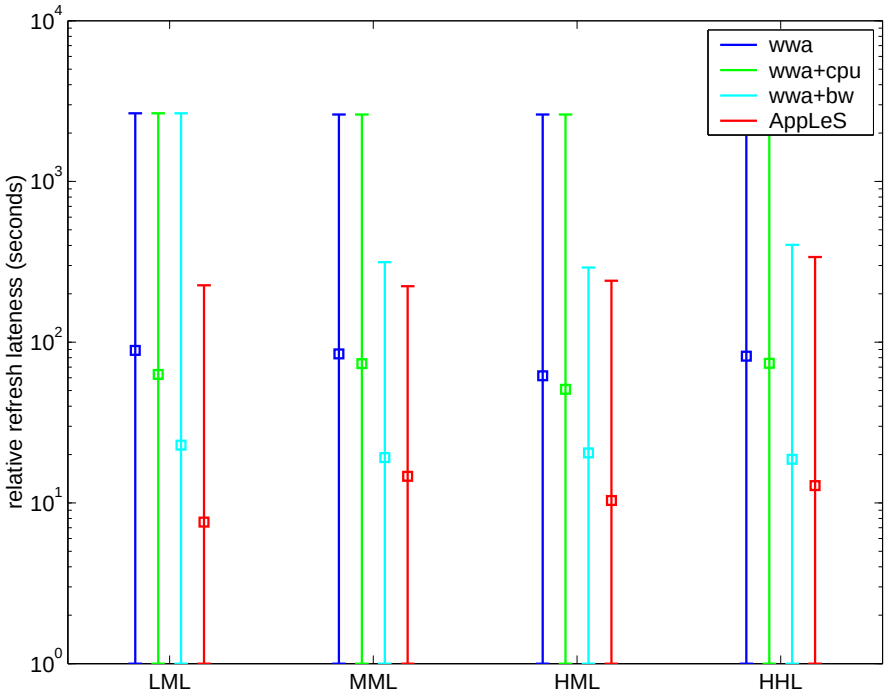


Figure IV.18: Simulation results grouped by partial order P_2. Each boxplot summarizes 40 simulations.

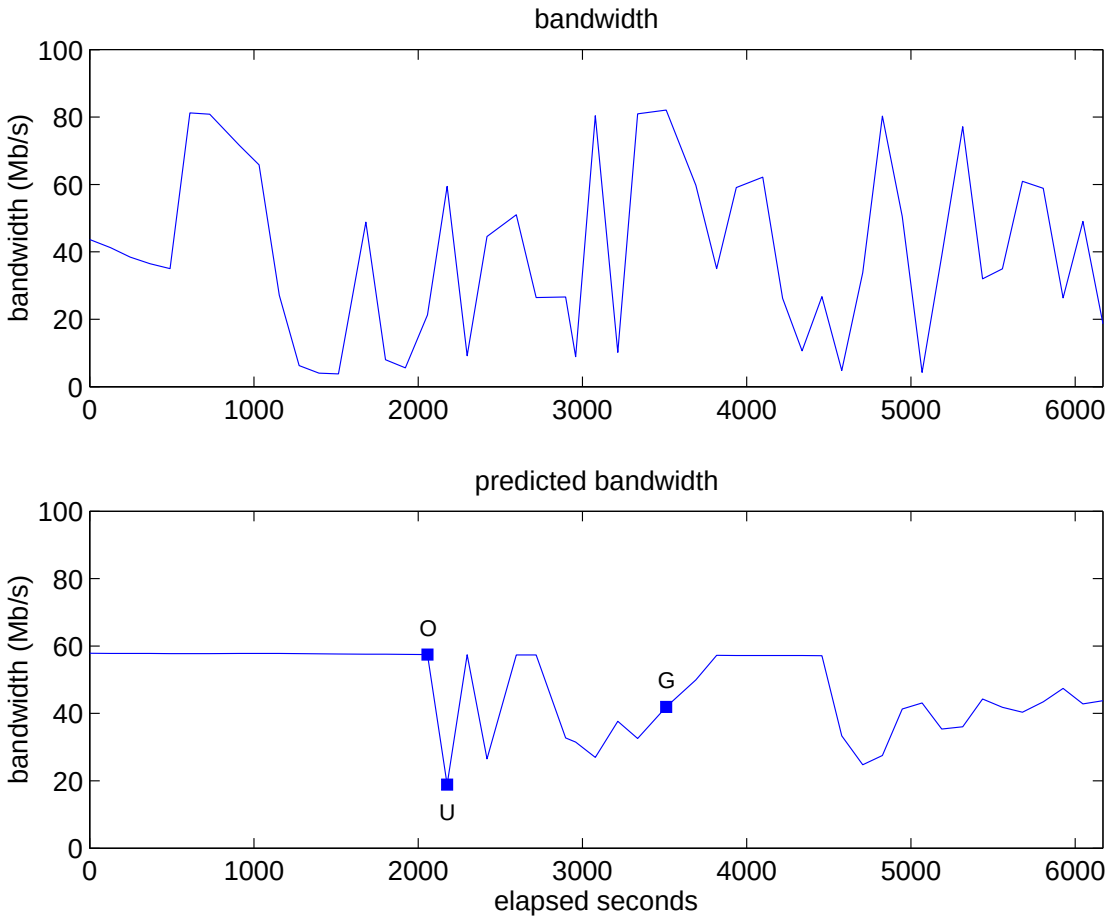


Figure IV.19: Bandwidth trace taken from torc8.cs.utk.edu to torc4.cs.utk.edu during the morning of February 12, 2001.

We emphasize that this is not a precise method for measuring Grid predictability but is one way to represent the results of all simulations. We describe our scoring technique below.

For a trace with predictability p, we arbitrarily assign it the weight γ using the following:

$$\gamma = \begin{cases} 1 & \text{if } p = L \\ 2 & \text{if } p = M \\ 3 & \text{if } p = H \end{cases} \tag{IV.12}$$

For each simulation with Grid type $p_1p_2p_3$, we assign it the *score*, Γ, using the following:

$$\Gamma = \sum_{i=1}^{3} \gamma(p_i) \tag{IV.13}$$

Note that the weighting scheme in Equation IV.12 results in the bound

$$3 \leq \Gamma \leq 9, \tag{IV.14}$$

where '3' indicates a Grid with low predictability and '9' indicates a Grid with high predictability. Next the simulation results are categorized into seven groups based on their score. The cumulative distribution functions of Δ_l for the simulations in each group are plotted in Figure IV.20. This figure clearly shows that the performance of the AppLeS scheduler increases as predictability increases. Here, we see that for Grids where $\Gamma = 9$, the AppLeS scheduler performs really well with over 90% of its refreshes having a Δ_l under 10 seconds. Similarly the AppLeS scheduler performs well when $\Gamma = 8$ and $\Gamma = 7$. Conversely, in Grids where $\Gamma = 3$, only about 40% of refreshes have a Δ_l under 10 seconds.

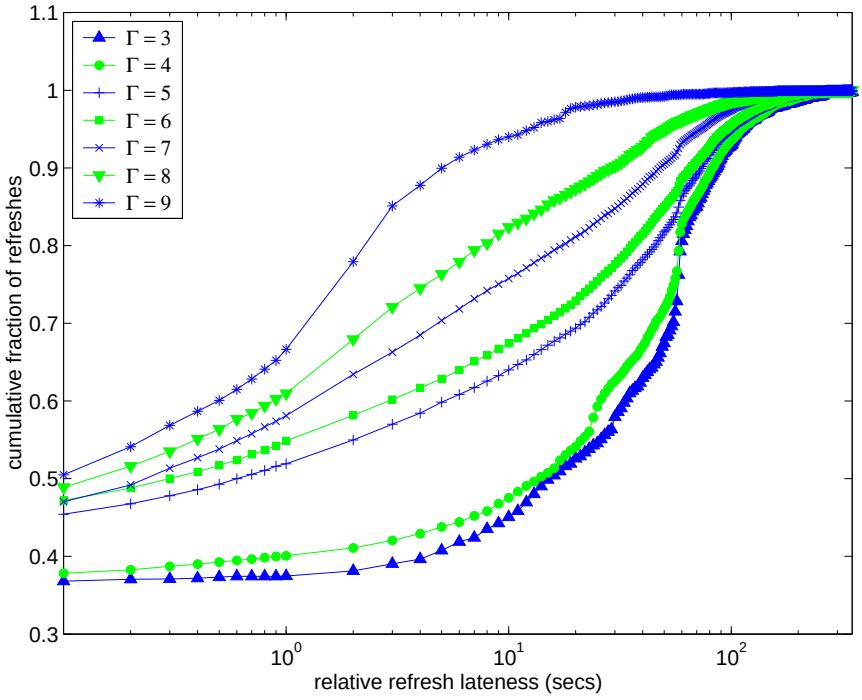


Figure IV.20: AppLeS cumulative distribution functions for Δ_l grouped by Γ.

IV.B.5 Summary

In this section, we studied the impact of dynamic load predictions on scheduler performance. We compared the AppLeS to three other schedulers which used no or partial dynamic information. We found that dynamic load predictions significantly improved real-time execution of on-line parallel tomography. In particular, we found that the performance gain was largely due to bandwidth predictions. We then examined the impact of bandwidth predictions on the performance of the AppLeS. Our experiments show that the performance of the AppLeS is largely dependent on the quality of bandwidth predictions.

IV.C Tunability Experiments

In Section I.C, we motivated the design of on-line parallel tomography as a tunable application for dynamic Grid environments. In this section, we as-

sess the usefulness of tunability; we say that tunability is *useful* if changing the configuration at run-time (from the previous configuration) results in a better configuration for the user and/or better real-time execution than not changing the configuration. We conduct a case study of tunability in Grids composed of two clusters of workstations and a supercomputer. These Grids are characterized by the variability of their traces as described further in Section IV.C.1. For each Grid, we study how the configuration of on-line parallel tomography would change for a user running back-to-back experiments during a two-day period. Section IV.C.2 describes the experiments and Section IV.C.3 describes the user model used for these experiments. The results described in Sections IV.C.4 and IV.C.5 show that application tunability was exploited frequently and therefore provide a case for tunability in dynamic Grid environments.

IV.C.1 Grid Construction

In order to have comparable results, we study a fixed Grid topology composed of a cluster of 8 workstations, a cluster of 16 workstations, and a supercomputer. Figure IV.21 illustrates the Grid topology. We then study tunability under different variability conditions. In these experiments, we look at the variability of both bandwidth and CPU availability traces. To collect traces for these experiments, we use a similar method as that described in Section IV.B.4.1. The difference is that we label our trace categories in terms of variability, i.e., the coefficient of variance is used as a coarse measurement of trace *variability* rather than predictability. A Grid can be described using a tuple, $(v_1, v_2, v_3, v_4, v_5)$, where $v_1, v_2, v_3, v_4, v_5 \in \{L, M, H\}$; v_1 is the type of bandwidth trace used for the network link between supercomputer and the writer, v_2 is the type of CPU availability traces used for the cluster of 8 workstations, v_3 is the type of bandwidth trace used for the network link between the cluster of 8 workstations and the writer, v_4 is is the type of CPU availability traces used for the cluster of 16 workstations, and v_5 is the type of bandwidth trace used for the network link between the cluster

of 16 workstations and the writer. For convenience, we abbreviate $low, medium$, and $high$ as L, M, and H respectively and write a tuple as $v_1v_2v_3v_4v_5$. There are a total of 243 different types of Grids.

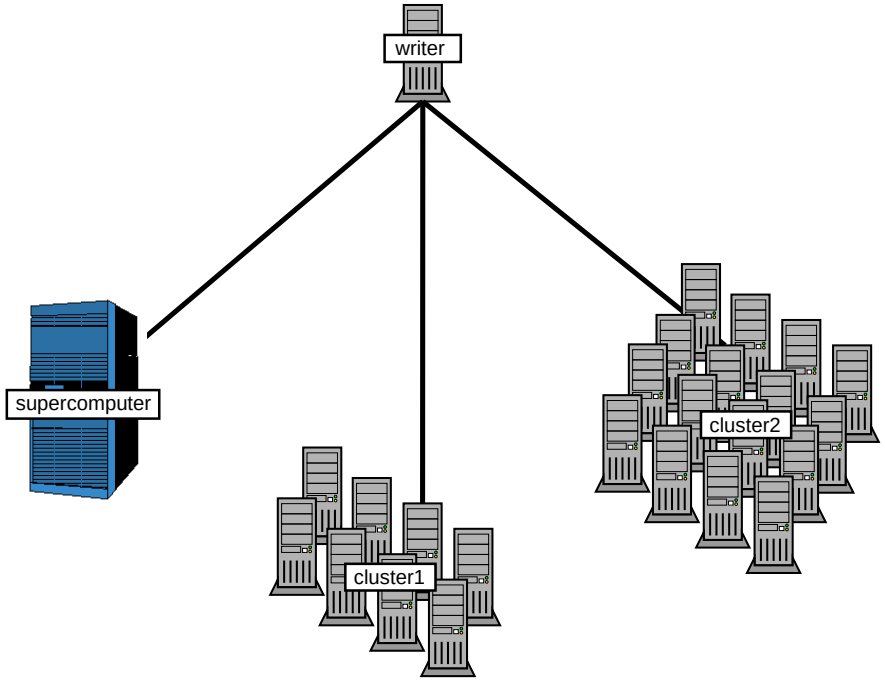


Figure IV.21: Grid topology for tunability experiments.

Note that to model the load on the supercomputer, we collected immediately available information from SDSC's Blue Horizon [3] using the Maui Scheduler's command showbf [33]; the trace was collected from February 9 to April 23, 2001 using a sample period of 5 minutes.

IV.C.2 Experiments

We consider two different on-line parallel tomography experiments:

$$E_1 = (45, 61, 1024, 1024, 300) \text{ and } E_2 = (45, 61, 2048, 2048, 600) \qquad \text{(IV.15)}$$

As described in Sections I.A and I.B, these two experiments are representative of the size of experiments run by NCMIR users and correspond to datasets collected from $1k \times 1k$ CCD camera and $2k \times 2k$ CCD camera respectively.

IV.C.3 User Model

In order to study the usefulness of tunability, we model how a user would choose a triple and then watch how it changes over time. For these experiments, we chose a simple user model. We assumed that the user would always choose triples that have the lowest f, followed by the lowest r. We also used the following charging model for service units:

$$su = n_m \times p \times a \tag{IV.16}$$

Since $p = 61$ and $a = 45$, su will always be a multiple of $61 \times 45 = 2745$.

The parameter bounds for the (45, 61, 1024, 1024, 300) experiment, are as follows:

$$\begin{aligned} 1 \leq f \leq 4 \\ 1 \leq r \leq 13 \\ 0 \leq su \leq 137250 \end{aligned} \tag{IV.17}$$

Similarly, for the $(61, 2048, 2048, 600)$, the bounds are:

$$\begin{aligned} 1 \leq f \leq 8 \\ 1 \leq r \leq 13 \\ 0 \leq su \leq 137250 \end{aligned} \tag{IV.18}$$

In both cases, the upper bound on su corresponds to 50 processors.

IV.C.4 Tunability Results

For each experiment, E_1 and E_2, we ran 243 simulations, one simulation for each of the 243 types of Grids. To simulate a user running back-to-back on-line parallel tomography experiments, we executed the scheduler every 45 minutes throughout the two-day period. For each two-day period, there were a total of 61 on-line parallel tomography experiments. Each time, we chose one triple according to the user model. There were a total of 14,823 on-line parallel tomography experiments for all 243 simulations.

In Figure IV.22, we display the range of triples found by the AppLeS scheduler for the E_1 experiments in a 3D graph. Each quadrant of Figure IV.22 displays a different view of the 3D graph. Here we see that most of the refresh factors fall within 1 and 3. Furthermore, no more than 3 processors are ever picked on the supercomputer. Similarly, we display the range of triples found for the E_2 experiments in Figure IV.23. Note, that since the projections are larger in this experiment, we can use a higher reduction factor. Here we can see that up to 25 processors are used on the supercomputer. Note on these types of Grids, it is not possible to get triples (1,1,x) or (1,2,x).

Now, we look at how the triples change within a single simulation. Suppose $T = \{1, ..., 61\}$ is a set of triples picked by the user during a 2-day simulation. For any $t_i, t_{i+1} \in T$, if $t_i \neq t_{i+1}$, then we say that the user's triple *changed*. We use the *number of changes* within a specified time period to measure the usefulness of tunability. For example, when the triple change frequency is low, we say that tunability is not useful. That is, it is likely that a user could use the same configuration from run to run and not experience a significant drop in performance. This was the case with the $MLLLL$ Grid and E_1; the user's triple remained constant at $(1, 1, 0)$ throughout the simulated 2-day period. Conversely, when the triple change frequency is high, we say that tunability is useful. We predict that a user running with the same configuration from run to run would experience significant performance drops and/or would under-utilize the resources. With E_2 the Grid,

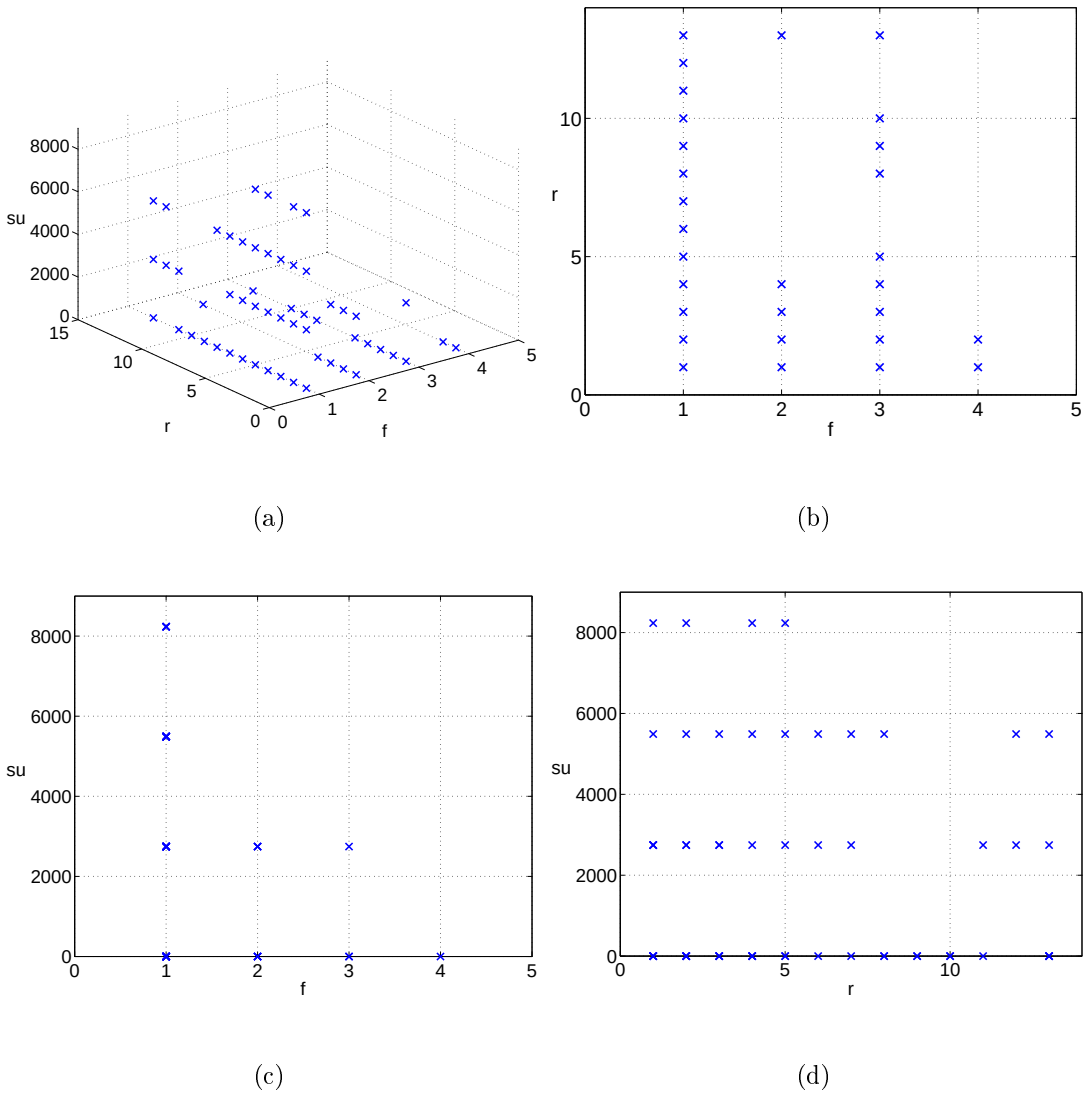


Figure IV.22: Triples found for $(61, 1024, 1024, 300)$ experiment.

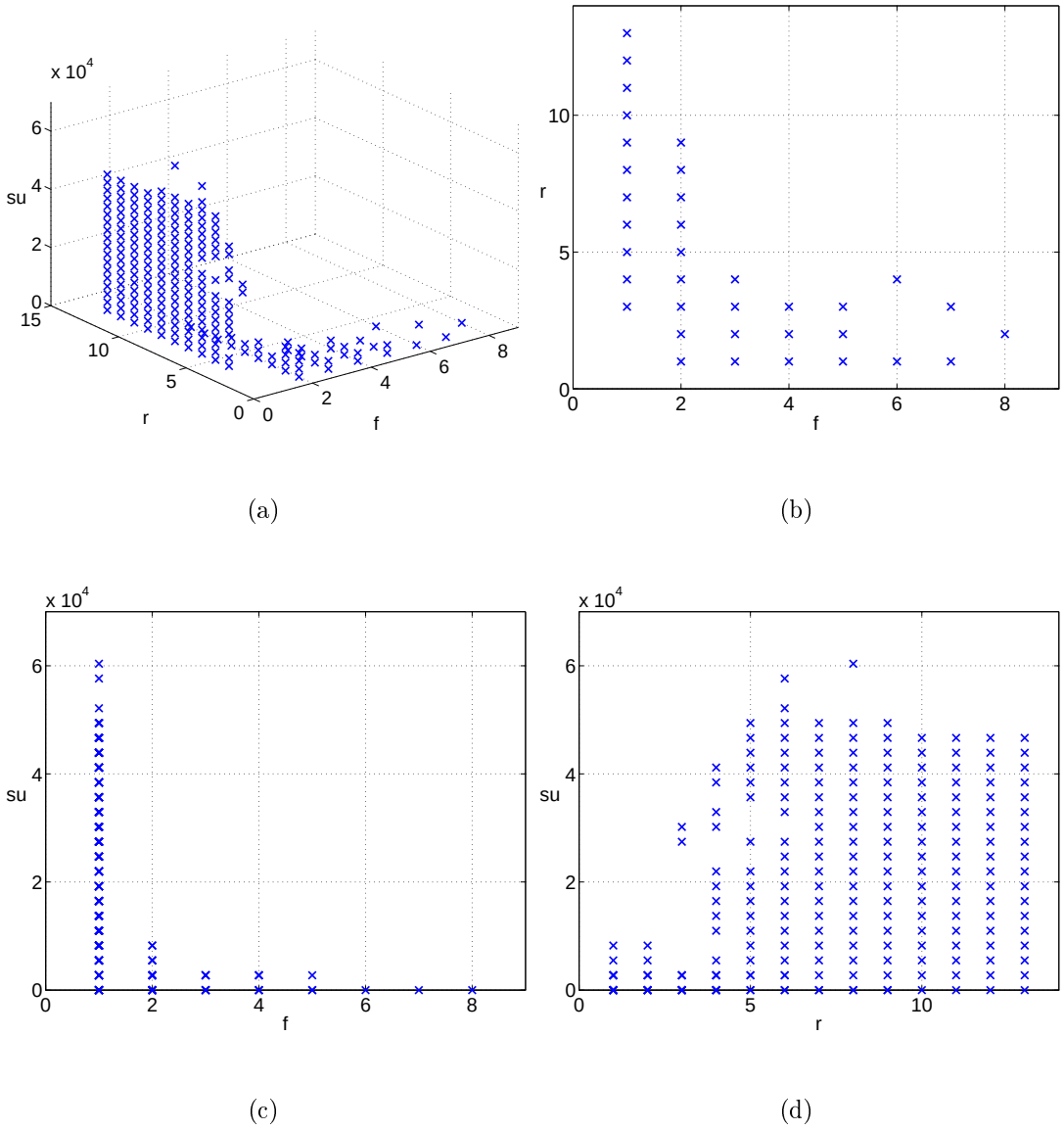


Figure IV.23: Triples found for $(61, 2048, 2048, 600)$ experiment.

$MLMMH$, exhibits this type of performance; the user's triple changed 44 times during the 2-day period. We show the user's triples for the $MLMMH$ Grid in Table A.1; it also shows the other triples the AppLeS scheduler found to be feasible.

Consider now the results of all simulations for both the E_1 and E_2 experiments. Using the user model outlined in Section IV.C.3, the triple changed 1910 out of 14823 times for the E_1 experiments and 3813 out of 14823 times for the E_2 experiments. Therefore, overall there was a 12.9% chance the triple changed from run to run for the E_1 experiments and 25.7% chance for the E_2 experiments. For each simulation, we also calculated how many times the parameters, f, r, and su changed over the 2-day period. The results for the E_1 experiments are displayed in Figure IV.24 and the results for the E_2 experiments are displayed in Figure IV.25.

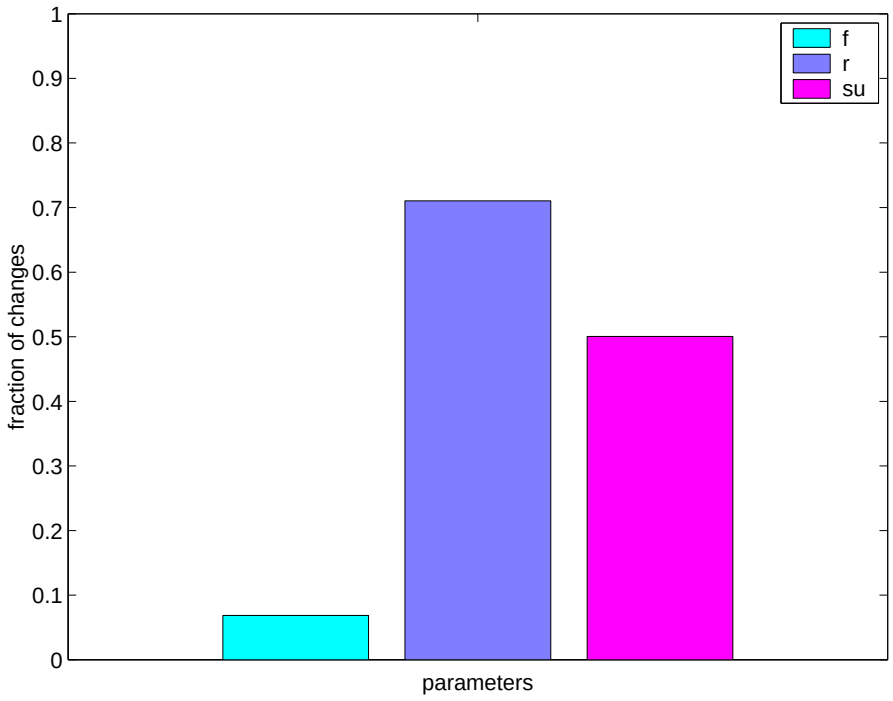


Figure IV.24: Frequency of parameter changes for E_1 experiments.

In both cases, r was the parameter that changed the most frequently, followed by su. Furthermore, we see that the frequency of change in reduction factor more than doubled from the E_1 to E_2 experiments.

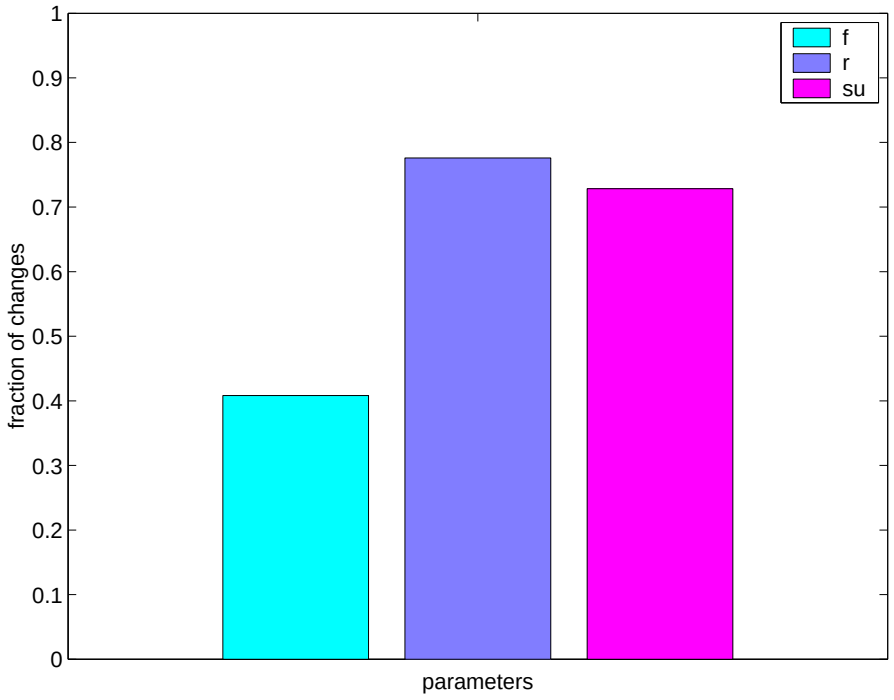


Figure IV.25: Frequency of parameter changes for E_2 experiments.

IV.C.5 Partial Order Results

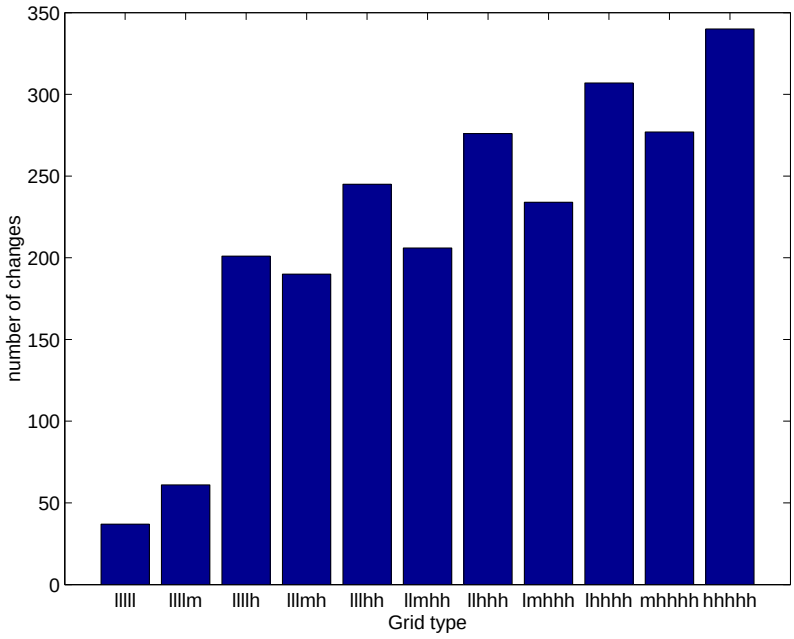
In this section, we study the relationship between frequency of triple changes and Grid variability. We picked 7 partially ordered Grids which increase in variability:

$$
\begin{array}{r}
LLLLL \\
LLLLM \\
LLLLH \\
LLLMH \\
LLLHH \\
LLMHH \\
LLHHH \\
LMHHH \\
LHHHH \\
MHHHH \\
HHHHH
\end{array}
$$

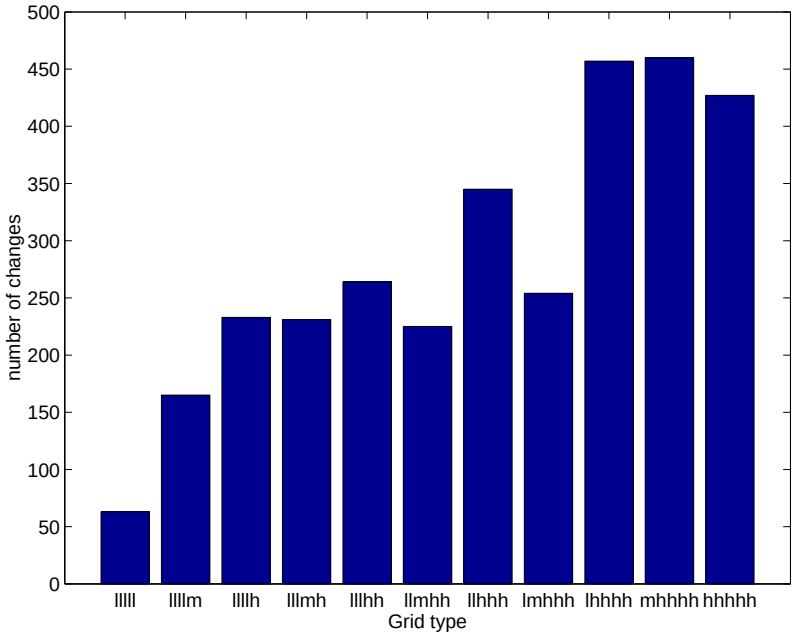
We then generated 20 different instantiations of each Grid type and simulated both E_1 and E_2 over a 2-day period. The results are displayed in Figures IV.26 and IV.26 respectively. These figures show that the frequency of triple changes does increases as Grid variability increases. However, we note that the increase in not monotonic.

IV.C.6 Summary

In this section, we did a case study on the usefulness of tunability in a fixed Grid topology composed of two workstation clusters and a supercomputer. We looked at tunability for two types of experiments representative of NCMIR users. We then ran simulations to study how the configuration of on-line parallel tomography would change for a user running back-to-back experiments during a two-day period. The goal was to measure the usefulness of tunability in Grids that differ in resource variability. We found that on average, there was a 12.9% likelihood that user's triple would change from one run to another for the $1k \times 1k$ experiments and 25.7% chance for the $2k \times 2k$ experiments. We also found that the usefulness of tunability increased as the variability of the Grid increased.



(a)



(b)

Figure IV.26: Partial order results: frequency of triple changes by (a) $1k \times 1k$ and (b) $2k \times 2k$

We conclude that tunability was useful in this fixed Grid topology and therefore provided a case for tunable on-line parallel tomography.

IV.D Scheduling Latency

In this section, we assess the AppLeS' scheduling latency. We define the *scheduling latency* to be the time it takes for the AppLeS scheduler to discover a set of feasible triples for an on-line parallel tomography experiment E and set of resources M. The scheduling latency is dependent on the size of the parameter space (see search algorithm in Figure III.2) and the execution time for the linear program solver.

We timed all experiments outlined in Section IV.C and grouped results by the type of experiment, E_1 and E_2. A histogram for the E_1 experiment search times is displayed in Figure IV.27 and a histogram for the E_2 experiment search times is displayed in Figure IV.28. From these results we see that for most experiments, the scheduling latency is nominal (88% of E_1 experiments and 63% of E_2 experiments had a second or less scheduling latency). Table IV.11 displays summary statistics for both E_1 and E_2 experiments. Here we see that the mean scheduling overhead is .35 seconds for the E_1 experiments and .99 seconds for the E_2 experiments. Therefore, the scheduler overhead more than doubled in time. This is warranted given that the parameter space for E_2 is larger than E_1's. (Recall that for E_2 the bound on f is between and 1 and 8 while the bound on f for E_1 is between 1 and 4).

Finally, there were a handful of outliers in both the E_1 and E_2 experiments that are too small to see on Figures IV.27 and IV.28. For the E_1 experiments, .09% of the experiments had search times between 3 and 8 seconds; for the E_2 experiments, .5% of the experiments had search times between 3 and 9 seconds. Due to time constraints, we were unable to determine the cause. However, we note that the percentage of these higher search times is nominal.

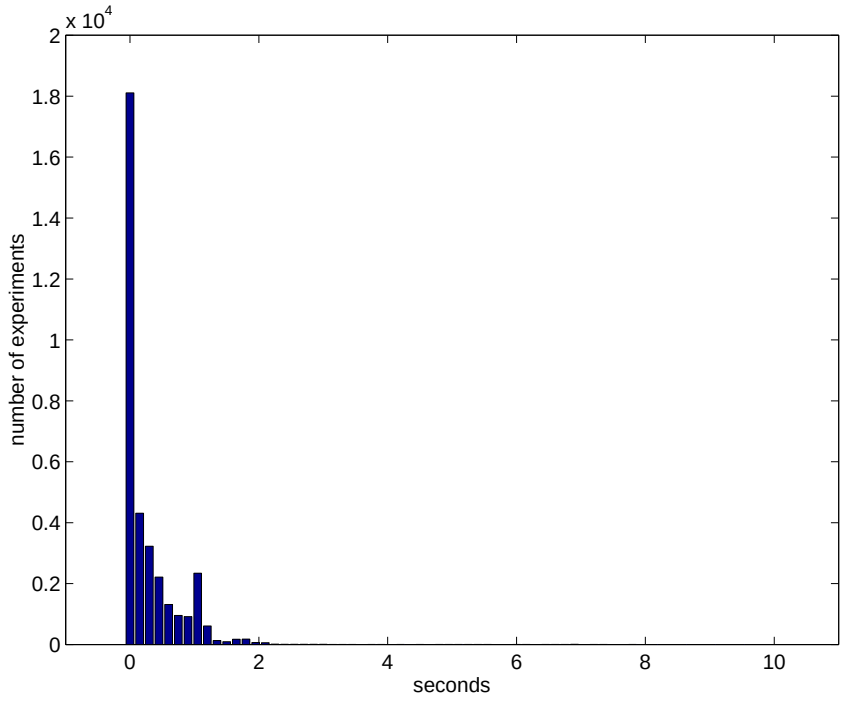


Figure IV.27: AppLeS scheduling latency for E_1 experiments.

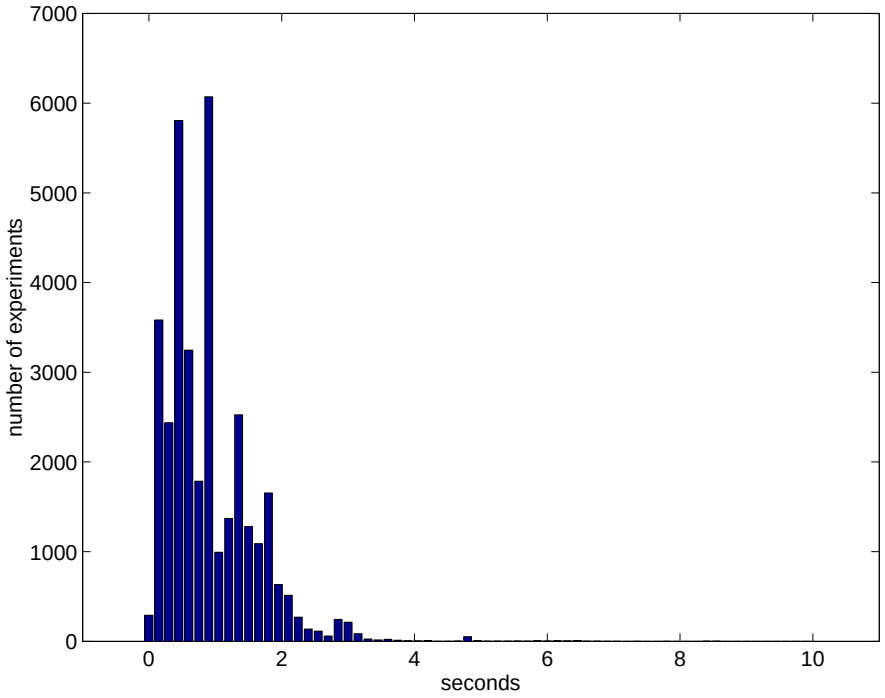


Figure IV.28: AppLeS scheduling latency for E_2 experiments.

	mean	std	min	max	median
E_1	0.35	0.45	0.04	7.85	0.13
E_2	0.99	0.68	0.11	8.68	0.92

Table IV.11: Summary statistics for AppLeS search times.

IV.D.1 Summary

To assess the impact of the AppLeS scheduler on application execution time, we studied the *scheduling latency* introduced by the AppLeS scheduler. We found that for the majority of executions, the AppLeS scheduler introduced a nominal scheduling latency of less than two seconds.

Chapter V

Related Work

On-line parallel tomography has also been addressed as part of the Computed Microtomography (CMT) project [52, 53]. Projections are collected from the Advanced Photon Source (APS) at Argonne National Laboratory, processed by an SGI Origin 2000, and visualized on an ImmersaDesk [15] or in a CAVE [12]. The CMT on-line parallel tomography code specifically targets high-speed networks and supercomputers and is a slightly extended version of the GTOMO code described in Section I.A.1[1]. The CMT extension enables data to be taken directly from APS and introduces processing stages. Each processing stage refills the work queue and results in a refresh to the tomogram. This is the same technique that was described in Section II.A where work is repeated in each stage. Thus, the on-line parallel tomography implementation presented in this thesis differs from CMT's in that it enables the R-weighted backprojection method to execute as an augmentable technique. Note that it would be straightforward to add the same extension to the CMT code in order to improve real-time execution. Second, our implementation enables on-line parallel tomography to execute across a more diverse set of resources (e.g. workstations, space-shared supercomputers, lower-capacity networks) through the use of application tunability.

[1]The base code for the CMT implementation of on-line parallel tomography and the base code for the implementation described in this thesis are the same. We refer to the base code as GTOMO in this thesis.

Application tunability is a concept that has been applied in the MILAN project [9] and in [17]. In MILAN, tunability is used by the system scheduler to improve throughput. The system scheduler is referred to as the *QoS arbitrator* and is responsible for allocating processors to application tasks. Each application has a *QoS agent* which interacts with the QoS arbitrator to ensure that its execution requirements are being satisfied. The QoS agent is automatically generated from annotated code. Our work differs from MILAN's in that our objective is to use tunability to improve *application* performance rather than system performance. We provide a single AppLeS process which functions as both the application's QoS agent and QoS arbitrator. While MILAN provides a simpler API, it is currently unable to sufficiently capture the requirements of on-line parallel tomography because the QoS arbitrator does not schedule bandwidth on network links. Given the large amount of data transfer required for on-line parallel tomography, the ability to express bandwidth requirements is critical to achieving real-time execution performance.

The work presented in [17] also uses tunability to improve application performance. Two applications are presented and classified as prediction-based, best effort, real-time applications. Using predictions of application performance based on dynamic load predictions, the application is mapped to a set of resources. Our work differs from theirs in that predictions of application performance are model-based rather than history-based.

Finally, the AppLeS described in this thesis builds upon other previous AppLeS work [49, 48, 16, 46] in its strategies for resource selection and work allocation. These AppLeS have focused on improving the performance of applications with *fixed* configurations. The AppLeS described herein distinguishes itself from these schedulers in its ability to improve the performance of an application (with multiple configurations) by exploiting its tunability.

Chapter VI

Conclusion

In this thesis, we implemented a Grid-enabled version of on-line parallel tomography which provides soft real-time feedback to users collecting data from a powerful electron microscope located at NCMIR. Acquiring data from NCMIR's microscope is a lengthy process and is susceptible to configuration errors. Soft real-time tomography feedback, which has been previously unavailable to NCMIR users, is important because it will allow users to quickly identify configuration problems and interact with the microscope in order to more efficiently acquire data from it. In this section, we summarize the contributions of the work for each chapter and conclude with future work.

In Chapter II, we motivated an extension to GTOMO to allow for on-line parallel tomography. This extension significantly reduced the amount of computation required for real-time execution of on-line parallel tomography by enabling the R-weighted backprojection method to execute as an augmentable technique. This required a change from a work queue scheduling strategy to static work allocation. This extension is more computationally efficient than adapting the off-line parallel tomography algorithm to on-line execution, but does not have the run-time adaptive scheduling advantage of work queue. We then defined a configuration of on-line parallel tomography as a triple, (f, r, su). These parameters represent resolution of the tomogram, frequency of refinements to the tomogram, and cost

of execution. These tunable parameters allow the application to be adapted to different resource availabilities.

In Chapter III, we defined a user-directed AppLeS. The AppLeS exploits the tunability of on-line parallel tomography to determine a schedule for soft real-time execution of the application over a set of resources at run-time. The scheduler utilizes user constraints, an application model based on soft deadlines, and dynamic resource load predictions to formulate multiple constrained optimization problems which are solved to determine feasible run-time configurations. We showed that each optimization problem could be efficiently and effectively solved using mixed-integer programming. The configurations are displayed as choices to the user where each configuration involves trade-offs between resolution of the tomogram, frequency of refreshes, and cost of execution. Once an appropriate configuration is chosen by the user, the scheduler selects resources, allocates work, and executes the application.

Finally in Chapter IV, we evaluated the impact of dynamic information on scheduler performance. We first ran experiments that simulated on-line parallel tomography at NCMIR. We found that the AppLeS achieved near perfect real-time execution when it used perfect load predictions. These results also showed that bandwidth predictions were the most significant factor to improving scheduler performance. We then ran experiments that simulated on-line parallel tomography at NCMIR with imperfect load predictions. These results indicated that the scheduler's performance was susceptible to the quality of the bandwidth predictions. Further experiments showed that scheduler performance degraded as the quality of bandwidth predictions degraded. Second, we ran a set of experiments where we examined the usefulness of tunability on Computational Grids. Our results showed that the usefulness of tunability increased as Grid variability increased. Finally, we showed that the scheduling latency introduced by the AppLeS was nominal.

Future work on this research would be to reduce the impact of bad predictions on real-time execution performance. One approach would be to extend

the AppLeS to reschedule the application during run-time (since our current static work allocation strategy does not perform run-time adaptive scheduling). This strategy would allow the application to better tolerate bad predictions by changing the work allocation during run-time. The first step would be to detect a need for rescheduling by weighing the potential benefit of rescheduling with the overhead of rescheduling such as in [45]. The AppLeS would then find a new work allocation using current dynamic resource load information. The new work allocation would be compared to the old work allocation to find an efficient way to shuffle the slices among ptomos.

A second way to reduce the impact of bad predictions would be to use a stochastic approach as outlined in [42]. In this work, NWS prediction error information was used to represent the load on a resource using a range of values. For on-line parallel tomography, the value we choose to represent the load on a resource could be based on how conservative the user wanted to be with their scheduling strategy. That is, a user could choose a more conservative, but possibly less efficient scheduling strategy or a less conservative, but possibly more efficient scheduling strategy. The user's conservativeness could be represented as a fourth parameter of the on-line parallel tomography configuration.

Finally, we would like to deploy the implementation of on-line parallel tomography described in this thesis into production at NCMIR. We expect that real-time feedback will allow NCMIR users to interact with the microscope to more effectively acquire data from the it. Overall, this will allow for more efficient usage of this powerful, scarce resource.

Appendix A

Tables

Table A.1: Feasible triples for highly variable Grid, MLMMH.

Time (s)	**Triple chosen**	**Other feasible triples**
0	(1, 11, 13725)	(1, 12, 10980), (1, 13, 8235), (2, 2, 0), (3, 1, 0)
2745	(1, 12, 10980)	(1, 13, 8235), (2, 2, 0), (2, 3, 0), (3, 1, 0)
5490	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
8235	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
10980	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
13725	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
16470	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
19215	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
21960	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
24705	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
27450	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
30195	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
32940	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
35685	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)

Continued on next page

Table A.1 – continued from previous page

Time (s)	Triple chosen	Other feasible triples
38430	(1, 13, 10980)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
41175	(1, 12, 13725)	(1, 13, 10980), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
43920	(1, 13, 10980)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
46665	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
49410	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
52155	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
54900	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
57645	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
60390	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
63135	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
65880	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
68625	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
71370	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
74115	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
76860	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
79605	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
82350	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
85095	(1, 12, 13725)	(1, 13, 10980), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
87840	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
90585	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
93330	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
96075	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
98820	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
101565	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
104310	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
107055	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)

Continued on next page

Table A.1 – continued from previous page

Time (s)	Triple chosen	Other feasible triples
109800	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
112545	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
115290	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
118035	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
120780	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
123525	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
126270	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
129015	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
131760	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
134505	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
137250	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
139995	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
142740	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
145485	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
148230	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
150975	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
153720	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
156465	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
159210	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
161955	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
164700	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)

Bibliography

[1] Francine Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. Application Level Scheduling on Distributed Heterogeneous Networks. In *Proceedings of Supercomputing 1996* , 1996.

[2] Dimitri P. Bertsekas. *Nonlinear Programming*, chapter 1, page 2. Athena Scientific, 1999.

[3] Blue Horizon User Guide at `http://www.npaci.edu/Horizon`.

[4] Stefan D. Bruda and Selim G. Akl. Real-Time Computation: A Formal Definition and its Applications. Technical Report 435, Queen's University, 2000.

[5] Henri Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid* , May 2001.

[6] Henri Casanova and Jack Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputing Applications and High Performance Computing*, 1996.

[7] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for Scheduling Parameter Sweep applications in Grid environments . In *Proceedings of the 9th Heterogenous Computing Workshop*, May 2000.

[8] Henri Casanova, Graziano Obertelli, Francine Berman, and Rich Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of the Supercomputing 2000*, 2000.

[9] Fangzhe Chang, Vijay Karamcheti, and Zvi Kedem. Exploiting Application Tunability for Efficient, Predictable Resource Management in Parallel and Distributed Systems. *Journal of Parallel and Distributed Computing*, 60:1420–1445, 2000.

[10] CHPC webpage at `http://www.chpc.utah.edu`.

[11] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 5, page 83. M.I.T. Press, Third edition, 1990.

[12] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. *ACM Computer Graphics*, 27(2):135–142, July 1993.

[13] CTC webpage at `http://www.tc.cornell.edu`.

[14] David E. Culler and Jaswinder Pal Singh. *Parallel Computer Architecture*, chapter 1, pages 60–61. Morgan Kaufmann Publishers, Inc., 1999.

[15] Marek Czernuszenko, Dave Pape, Daniel Sandin, Tom DeFanti, Gregory L. Dawe, and Maxine D. Brown. The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays. *Computer Graphics*, 31(2):46–49, 1997.

[16] Holly Dail, Graziano Obertelli, Francine Berman, Rich Wolski, and Andrew Grimshaw. Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem. In *Proceedings of the 9th Heterogenous Computing Workshop*, May 2000.

[17] Peter A. Dinda, Bruce Lowekamp, Loukas Kallivokas, and David R. O'Hallaron. The Case for Prediction-based Best-effort Real-time Systems . Technical Report CMU-CS-98-174, Carnegie Mellon University, 1999.

[18] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. To be published in Intl. J. Supercomputer Applications, 2001.

[19] Ian Foster and Carl Kesselman. The Globus Project: A Status Report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998.

[20] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 12. Morgan Kaufmann Publishers, Inc., 1999.

[21] J. Frank and M. Radermacher. Three-Dimensional Reconstruction of Non-periodic Macromolecular Assemblies from Electron Micrographs . In J. K. Koehler, editor, *Advanced Techniques in Biological Electon Microscopy III*. Springer-Verlag, 1986.

[22] P. Gilbert. Iterative Methods for the Three-dimensional Reconstruction of an Object from Projections . *J. Theoret. Biol.*, 36:105–117, 1972.

[23] R. Gordon, R. Bender, and G.T. Herman. Algebraic Reconstruction Techniques (ART) for Three-dimensional Electron Microscopy and X-ray Photography . *J. Theoret. Biol.*, 29:471–481, 1970.

[24] A. Grimshaw, A. Ferrari, F.C. Knabe, and M. Humphrey. Wide-Area Computing: Resource Sharing on a Large Scale. *IEEE Computer*, 32(5), May 1999.

[25] M. Hadida-Hassan, S.J. Young, S.T. Peltier, M. Wong, S. Lamont, and M.H. Ellisman. Web-based Telemicroscopy. *J. Struc. Biology*, 125:235–245, 1999.

[26] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[27] A. C. Kak and M. Slaney. *Principles of Computerized Tomography Imaging*. IEEE Press, 1998.

[28] Reinhard Klette and Piero Zamperoni. *Handbook of Image Processing Operators*, chapter 4, pages 120–125. John Wiley and Sons, Ltd., 1996.

[29] Linear Programming FAQ webpage at `http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html`.

[30] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor—A Hunter of Idle Workstations. In *Proc. of the 8th Int'l Conf. on Distributed Computing Systems*, pages 104–111, 1988.

[31] Jane W.S. Liu. *Real-Time Systems*, chapter 2, pages 26–33. Prentice-Hall, Inc., 2000.

[32] lp_solve FTP site at `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.

[33] Maui Scheduler webpage at `http://www.mhpcc.edu/ maui`.

[34] Robert Dant - MHPCC (personal communication, Jan 02, 2001).

[35] NCSA webpage at `http://www.ncsa.uiuc.edu`.

[36] Nonlinear Programming FAQ webpage at `http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html`.

[37] NPACI webpage at `http://www.npaci.edu`.

[38] G.A. Perkins, C.W. Renken, J.Y. Song, T.G. Frey, S.J. Young, S. Lamont, M.E. Martone, S. Lindsey, and M.H. Ellisman. Electron Tomography of Large, Multicomponent Biological Structures. *Journal of Structural Biology*, 120:219–227, 1997.

[39] G.A. Perkins, C.W. Renken, S.J. Young, S.P. Lamont, M.E. Martone, S. Lindsey, T.G Frey, and M.H. Ellisman. Electron tomography of large multicomponent biological structures. *J. Struct.Biol.*, 120:219–227, 1997.

[40] Radia Perlman. *Interconnections*, chapter 2, page 19. Addison Wesley Longman, Inc., second edition, 2000.

[41] M. Radermacher. Three-dimensional reconstruction of single particles from random and nonrandom tilt series. *J. Electron Microsc. Tech.*, 9:359–394, 1988.

[42] J. Schopf. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters*. PhD thesis, University of California, San Diego, 1998.

[43] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA)*, pages 39–48, February 1996.

[44] Gary Shao, Fran Berman, and Rich Wolski. Using Effective Network Views to Promote Distributed Application Performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications* , 1999.

[45] Gary Shao, Rich Wolski, and Fran Berman. Predicting the Cost of Redistribution in Scheduling. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing* , 1997.

[46] Shava Smallen, Walfredo Cirne, Jaime Frey, Francine Berman, Rich Wolski, Mei-Hui Su, Carl Kesselman, Steve Young, and Mark Ellisman. Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience. In *Proceedings of the 9th Heterogenous Computing Workshop*, May 2000.

[47] Gabriel E. Soto, Stephen J. Young, Maryann E. Martone, Thomas J. Deerinck, Stephan Lamont, Bridget O. Carragher, Kiyoshi Hamma, and Mark H. Ellisman. Serial section electron tomography: A method for three-dimensional reconstruction of large structures. *Neuroimage*, 1:230–243, 1994.

[48] Neil Spring and Rich Wolski. Application Level Scheduling of Gene Sequence Comparison on Metacomputers. *12th ACM International Conference on Supercomputing* , July 1998.

[49] Alan Su, Francine Berman, Richard Wolski, and Michelle Mills Strout. Using AppLeS to Schedule Simple SARA on the Computational Grid. *International Journal of High Performance Computing Applications* , 13(3):253–262, 1999.

[50] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing*, 1999.

[51] Andrew S. Tanenbaum. *Computer Networks*, chapter 1, page 8. Prentice Hall, Inc., Third edition, 1996.

[52] Gregor von Laszewski, Mei-Hui Su, Joseph Insley, Ian Foster, John Bresnahan, Carl Kesselman, Marcus Thiebaux, Mark Rivers, Steve Wang, Brian Tieman, and Ian McNulty. Real-time analysis, visualization, and steering of tomography experiments at photon sources. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, Apr 1999.

[53] Yuxin Wang, Francesco De Carlo, Ian Foster, Joseph Insley, Carl Kesselman, Peter Lane, Gregor von Laszewski, Derrick Mancini, Ian McNulty, Mei-Hui Su, and Brian Tieman. A quasi-realtime xray microtomography system at the Advanced Photon Source. In *Proceedings of SPIE*, volume 3772, 1999.

[54] Rich Wolski. Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, August 1997.

[55] Rich Wolski, Neil Spring, and Chris Peterson. Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service. In *Proceedings of Supercomputing 1997* , 1997.

[56] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *The Journal of Future Generation Computing Systems* , 1999.