Chapter 2

Peach: A simple Perl-based system for distributed computation and its application to cryo-EM data processing

Peter A. Leong[1#], J. Bernard Heymann[2#], and Grant J. Jensen[3*]

[1]Department of Applied Physics, California Institute of Technology, 1200 E. California Blvd., Pasadena, California 91125

[2]Laboratory of Structural Biology Research, National Institute of Arthritis, Musculoskeletal and Skin Diseases, National Institutes of Health, Bethesda, Maryland 20892

[3]Division of Biology, California Institute of Technology, 1200 E. California Blvd., Pasadena, California 91125

[#] These authors contributed equally

* Corresponding author, email address **jensen@caltech.edu**, 626-395-8827

2.1 Summary

A simple distributed processing system named "Peach" was developed to meet the rising computational demands of modern structural biology (and other) laboratories without additional expense by using existing hardware resources more efficiently. A central server distributes jobs to idle workstations in such a way that each computer is used maximally, but without disturbing intermittent interactive users. As compared to other distributed systems, Peach is simple, easy to install, easy to administer, easy to use, scalable, and robust. While it was designed to queue and distribute large numbers of small tasks to participating computers, it can also be used to send single jobs automatically to the fastest currently available computer and/or survey the activity of an entire laboratory's computers. Tests of robustness and scalability are reported, as are three specific electron cryomicroscopy applications where Peach enabled projects that would not otherwise have been feasible without an expensive, dedicated cluster.

2.2 Introduction

The availability of ever-faster computers continues to open new possibilities throughout science and in structural biology in particular. This leads us to plan increasingly demanding projects and gather the computational resources needed. In many structural biology laboratories, the mixtures of heterogeneous workstations purchased individually or in small sets for laboratory personnel in recent years constitute a wealth of underutilized capacity. Here we report the development of a Perl-based package called "Peach" that efficiently distributes computational tasks across such workstations without disturbing interactive users.

The motivation for this work arose out of our own structural biological studies in electron cryomicroscopy (cryo-EM). Modern cryo-EM has three distinct modalities: (1) "two-dimensional crystallography", in which many crystals of a specimen only a single unit cell thick are imaged at various tilt angles with respect to the beam; (2) "single particle analysis," in which thousands of fields of randomly oriented particles are imaged in projection; and (3) "tomography," in which a single, unique object is imaged iteratively while being incrementally tilted about some axis. In each case, the resulting images are merged to produce a three-dimensional reconstruction of the specimen, and the process involves a large number of small, easily separable, independent calculations (for recent reviews and some descriptions of the computational challenges in this field, see ((Walz and Grigorieff 1998; van Heel, Gowen et al. 2000; Fernandez, Lawrence et al. 2002; Frank 2002; Sali, Glaeser et al. 2003; Frangakis and Forster 2004; Orlova and Saibil 2004; Subramaniam and Milne 2004)). Glaeser has presented a "straw man argument" stating that solving the structure of a large protein complex by single particle analysis to near-atomic resolution with current algorithms would take even a state-of-the-art teraflop computer something like a year (Glaeser 1999). Even though we are still far away from this resolution goal for various reasons, in a typical cryo-EM laboratory today, computer power is already at a premium, and represents a real limitation. Researchers lose time logging in to multiple computers, manually distributing jobs across computers with different operating systems, generating custom scripts to submit jobs one after another through the night or weekend, watching for their completion, and coordinating computer usage with laboratory colleagues. Despite such efforts, most workstations are still only used to a small fraction of their capacity due to the difficulty of manually managing multiple tasks on multiple workstations.

To improve this situation, we searched for an inexpensive system to distribute jobs efficiently, easily, and securely across our set of workstations. Only a few options were available, including Open PBS from Veridian Systems and Condor (Tannenbaum and Litzkow 1995). While we have a running version of Open PBS on our Linux cluster, it has no "desktop harvesting", or in other words, it was not designed to take advantage of unused time on interactive workstations, as we desired. We downloaded and installed Condor, but disliked its complexity, as it required installation of separate executables for each platform, a large number of different types of daemons, over twenty-five different programs, and special submission description files. Further, source code was not available and the documentation warned of known security issues. Another well-known package is BOINC, the Berkeley Open Infrastructure for Network Computing, which mediates the SETI@home project (Anderson, Cobb et al. 2002). BOINC was designed for "public-resource" (as opposed to in-house, or "grid") computing, in which participants are random individuals who donate time on their personal computers, connected to the internet via telephone or cable modems or DSL. While wonderful for certain applications, BOINC would not be attractive for structural biological applications because of the large amounts of data needing to be transferred, the need for accuracy (which in public-resource systems is achieved by redundant computing or some kind of post-verification), and the large amounts of memory often required.

Not finding a suitable alternative, we developed Peach, a simple Perl-based distributed computation system. The small number of scripts that constitute the system are easy to install and require no compilation. Peach is easy to use, easy to administer, free, robust, scalable, secure, and immediately compatible with almost any Unix operating system and non-interactive executable. We have installed it in two laboratories, where it has accelerated routine work and brought several structural biology projects to success that would not otherwise have been feasible without the purchase of an expensive, dedicated computer cluster.

2.3 Design

2.3.1 Design Philosophy — From the user's point of view, the goal was to develop a system that would accept anywhere from one to thousands of jobs and automatically process them as fast as possible using the existing workstations in the laboratory, but without disturbing interactive users. Two scenarios were envisioned: (1) when one or more workstations were idle, in which case a new job would be sent immediately to the fastest one suitable, and (2) when all workstations were busy, in which case submitted jobs would be queued and distributed later. Further, the system needed to be simple to use and administer, scalable, secure, robust, and as compatible with the existing hardware and software in structural biology as possible.

2.3.2 Implementation — Peach was implemented following a client-server model, in which a single job "server" daemon runs on one workstation and maintains a queue of jobs to be done, while job "client" daemons run on all the other workstations, periodically reporting their state and running the jobs assigned to them. Three simple "access" clients constitute the complete user interface: (1) *psubmit*, which when given any executable file with flags and options as arguments, submits that job to the system; (2) *pview*, which generates reports on the status of the participating computers and submitted jobs; and (3) *pkill*, which terminates and/or removes jobs. Only clients initiate communication, so new clients can join and others terminate without disrupting the server.

2.3.3 Information Flow — Work begins when a user submits a job with the *psubmit* access client. *psubmit* writes an "execution script" on a shared disk which contains paths to an appropriate executable for each participating operating system. Next the *psubmit* client sends a message to the job server with the name of the execution script and the identity of the user, plus optional information about preferred processors and email addresses for reporting. The job server stores this information in memory and on disk. Meanwhile the job clients on all the participating workstations periodically report their status to the job server. When the job server has a job in the queue and a suitable processor is reporting that it is idle, the server responds to the corresponding client with the name and path of the execution script. The job client, which is owned by root, forks a child process whose ownership is changed to the submitting user. Then the child process runs the execution script with "niced" priority (i.e., a low priority to allow other, interactive users better access) and writes the standard output and error files. After the job terminates, the job server sends an e-mail message to the user if requested. If during execution an interactive user begins to use the console, the job client immediately suspends active Peach jobs for a configurable time period. If that period will exceed the time the job had already been processing, the job client "releases" the job back to the job server for reassignment elsewhere. If a process fails (returns a non-zero value to the operating system), it is reassigned, but if it fails again, it is removed from the queue and the owner is notified.

2.3.4 The Job Server — The job server acts as a job broker, storing information about all the submitted jobs and processors participating in the system and matching them efficiently. The server also writes state and log files about transactions, job completions, and job failures. In the event the server crashes, it can be easily restarted (or even automatically restarted if desired) on any workstation, where it will read the state files and proceed without affecting current or waiting jobs. Clients automatically find the new IP address and port number of the server in the configuration file on the shared disk. The server has several built-in mechanisms to handle unexpected states appropriately.

2.3.5 The Job Clients — Each of the participating computers (regardless of the number of processors on the computer) has exactly one job client running at all times, which cycles automatically every few seconds to (1) monitor processor usage, (2) gather information about the status of current jobs, (3) suspend active jobs if a user begins interactive use at the console, (4) make decisions about whether to "release" suspended jobs back to the server, (5) report to the job server, and (6) launch newly assigned jobs from the server.

2.3.6 Use of Existing Capabilities — The system was written in Perl, which is installed by default on almost all Unix-variants. It makes use of only standard components available in recent distributions (Perl 5.8, March 2000, or later). This ensures cross-platform compatibility and ease of installation, since no compilation is required. For simplicity, data exchange across platforms is managed through existing TCP/IP and NFS services by mounting on all participating computers at least one shared disk where the Peach scripts, some configuration/state files, executables for each platform, and data are located. Additional shared data disks can be mounted on some or all of the participating computers (Figure 2-1). In this way large data files are not copied, even temporarily, to local disks, but rather are read from and written to a shared, central disk system. All messages are passed in standard XML format to increase compatibility with other software and in anticipation of future developments.

2.3.7 Security — Administrators and users are registered with password-protected accounts internal to Peach, allowing users access to all the participating computers without the requirement of accounts for all the users on all the computers. Each client (such as *psubmit* or *pview*) requires a valid username and password. Messages carry unique signatures formed through a digest (a transformation of the text such that it cannot be decoded and read) of the username, the password, the message itself, and a unique authorization string provided by the job server. The recipient verifies that signature using it's knowledge of the unique string and its own database of registered users and passwords, preventing unauthorized messages. Finally, while Peach job clients are owned by root, the ownership of their child processes which actually launch all the jobs are changed to the submitter, and no jobs are allowed to run as root. Thus damage from poorly designed or malicious jobs is limited to the submitting account.

2.3.8 Peach Administration — A primary design goal of Peach was that it be simple, both for users and for the administrator. To install Peach, a simple script copies all the required files (seven executable Perl scripts and seven supporting files) to a program directory on a shared disk that must be available to all participating computers. No programs have to be recompiled: any existing Unix command, script, or program can be immediately submitted as a job. The only requirement is that these programs are available, either as common utilities on all computers, or more typically, as executables on a shared disk. During setup, the job server and job client daemons must be launched and user accounts with names and passwords must be established. New shared disks and workstations can be added easily. The appropriate configuration files are generated during installation, but various parameters can be specially configured if desired. Typical Unix conventions for file locations and configuration have been followed as far as possible to facilitate administration.

Peach was designed to have robust, independent modules. Thus if the job server dies, it can be restarted on any other machine without disruption to current or waiting jobs. If a job client dies (for instance if a workstation is rebooted), there is no impact on any other client, and a new job client can be re-started and join the system at any time. If network delays slow communication, or a job client stops reporting for any reason, Peach self-recovers as soon as conditions improve. Peach does depend on a commonly shared disk for access to programs and data, however, which is a limitation we accepted to keep the system simple and avoid the complexities of copying large amounts of data across a network.

2.4 Tests and Results

2.4.1 Installation and Test Environments — Peach has been installed and tested now in two separate laboratories at the California Institute of Technology (Caltech) and the National Institutes of Health (NIH). At Caltech it was developed on an existing heterogeneous set of 17 computers including four Macintosh dual-G5s (2.0 GHz, 2.5 GB RAM), 12 PCs running Linux (2.2–2.4 GHz, 1.0–4.0 GB RAM, 1–4 processors each) and 1 SGI Fuel (0.6 GHz, 2.0 GB RAM). At the NIH, Peach distributes jobs to 11 heterogeneous computers, including 6 Macintosh dual-G5s (2.0 GHz, 5 GB RAM), 2 dual-MIPSpro SGI Octanes (0.25 GHz, 1 GB RAM) and 3 HP Alphas (0.7 GHz, 1.5–5 GB RAM, 1–4 processors). In both laboratories, a central shared disk was available to all the participating workstations, but various additional shared "data" disks came on- and off-line during the testing period. The local networks supported 1 Gbit/s Ethernet for communication and typically performed at 5–10% of nominal capacity. The Bsoft package (Heymann 2001) used for image processing was installed on the central shared disk with compiled versions for each operating system located in different directories. Peach was developed in the short span of a few months at Caltech within a network and computer setup configured for its use. The installation at the NIH, however, represented a useful test of how readily usable Peach would be by other groups whose hardware was not set up specifically for it. The main hurdles at the NIH were to arrange for a central disk that all the computers could access and to make all the users' individual and group identification numbers consistent across the set of participating computers. Further configuration entailed compiling all the required executables for image processing for the different platforms and installing those on the shared disk. After that, Peach was installed and configured in less than an hour.

2.4.2 Cryo-EM Applications — Peach has now been used for several of our electron cryomicroscopy projects. Three examples will be described. We have recently explored the potential benefit of cooling frozen-hydrated samples with liquid helium instead of liquid nitrogen in the context of electron cryotomography. In one test we recorded full tilt series of fields of a purified protein complex, the molluscan hemocyanin from Megathura crenulata, with total doses ranging from 10 to 300 electrons/Å^2, at each of the two temperatures. From each tilt series a three-dimensional reconstruction ("tomogram") of the field of particles was calculated, and individual hemocyanin molecules were manually identified. To measure the overall quality of the tomograms at each dose and temperature, approximately 100 hemocyanin molecules were aligned to the known 12 Å structure (Mouche, Zhu et al. 2003) using the program bfind (Bsoft) (Figure 2-2). Thus a three-dimensional translation and orientation search was performed for ~ 1,400 cube-shaped volumes of 64^3 voxels each.

In a second, related example, we recorded multiple, iterative images of fields of frozen hemocyanin particles using 10 electrons/$Å^2$ for each image. As more and more images were recorded, the structure of the particles degraded due to radiation damage. We measured the rate of degradation by picking 100 hemocyanin particles out of each image in the series and using them to calculate a three-dimensional reconstruction, which was compared to the known higher-resolution structure. By recording such "dose series" of many fields cooled by either liquid nitrogen or liquid helium and plotting the resolution of the resulting reconstructions as a function of dose, we were able to test whether deeper cooling with liquid helium delayed radiation damage as hoped (data not shown). We used Peach throughout this project to manage the literally hundreds of "single-particle" reconstructions involved. During a 23-day period a total of 2,146 jobs related to these hemocyanin projects were run on Peach, using 322 days of CPU time. This accounts for approximately 80% of the capacity of our 17 workstations during those days, all obtained without disturbance to the intermittent interactive users.

As a third example, we have simulated images of protein complexes embedded in vitreous ice under different imaging conditions using the so-called "multi-slice" algorithm (Cowley and Moodie 1957). Three-dimensional reconstructions were calculated with various alignment errors to explore their effect on resolution. The most computationally intensive part of this work is the atom-by-atom calculation of the atomic potential of each simulated cube of water and protein. In one recent batch of simulations, we used Peach to manage the calculation of 1947 images over a period of 3.6 days, logging 96 days of actual CPU time (Figure 2-3).

2.4.3 Robustness — The most common computer failures in our experience are stalled computers, disk problems, and network delays. Peach was designed to be as tolerant of these disruptions as practically possible, and several robustness tests were performed. In the first test, the job server daemon was terminated while managing a long queue of active and waiting jobs, as would happen, for instance, if the workstation hosting the job server hung or had to be rebooted. Active jobs continued without disturbance and began completing successfully. After two hours, the job server was restarted on its original host computer, and all the job clients re-initiated communications and began reporting and/or receiving new jobs as normal. In the second test, the job server daemon was again terminated while managing a long queue, but this time it was restarted on a different host computer. Again, no delays or complications were experienced, as the existing job clients and future access clients all found the new IP address and port number of the server from the configuration file and proceeded as normal. For the third test, a job client daemon was terminated. As expected, it was first listed by *pview* as missing, and then after one hour it was removed from the list of job clients and the jobs that had been assigned to it were re-queued and later distributed to other machines.

Without specific tests, we have observed the behavior of Peach under other challenging conditions. During periods of network delays, job clients were unable to report punctually to the server. This had little consequence, however, since active jobs continued running and only the brief breaks between jobs were extended. Of course data transfer to and from the shared disk was also delayed by network slowdowns, so network reliability and speed are areas for improvement. In one instance the central shared disk was inaccessible for several minutes, but normal communication and file transfer resumed once the disk became accessible again.

2.4.4 Scalability — It is important that distributed computation systems such as Peach maintain efficiency if more processors are added. Because Peach only distributes completely independent jobs, rather than interdependent parts of single jobs, the main bottleneck that arises when more processors are introduced is the response of the job server to each job client's report. Bottlenecks can also arise in accessing shared disks, but there is no explicit limit to the number of shared data disks that can be added to the system. While only one job client was intended to ever be running on any given computer, in order to explore Peach's scalability with our present hardware, we ran tests in which progressively larger numbers of job clients were added to the system by simply launching additional job clients on one of six chosen workstations at the rate of one additional client per minute. The corresponding server response times are plotted in Figure 2-4 for various settings of the job client reporting interval (the configurable time between when a job client receives a response from the job server and when that job client initiates its next report). For each reporting interval, the plot shows three distinct regions. Initially, the job server is unsaturated and responds immediately to all job clients. As the number of reporting clients increases, eventually the socket queues begin to fill, and the response time increases linearly. Because the server can no longer respond to the job clients' reports as fast as they come, one might expect the socket queues and therefore the response time to lengthen steadily, even in between the additions of new clients. What was observed, however, was that a new, stable response time was reached after each additional client entered the system. This happened because job clients do not initiate a new report until *after* they receive a response. Thus new reports replace old ones on the socket queue only as fast as the old ones are served with a response. This equilibrium becomes impossible, however, in the third region, after so many job clients are added that the number of reports waiting in the queue exceeds the number of connections available (a parameter set in the operating system kernel), and reports start to be refused. Thus with our current configuration of hardware and the default five-second job client reporting interval, Peach's job server can manage up to approximately 200 participating computers reliably. Arbitrarily larger clusters can be serviced simply by increasing the reporting interval appropriately in the main Peach configuration file.

2.5 Discussion

Among large computational tasks in structural biology (as well as all science), some are not easily separated into small, independent tasks. Instead, these require intensive communication between processes and rely on large, homogeneous clusters (so-called "supercomputers") that optimize inter-node communication speeds. There are also, however, a vast number of tasks which are trivially parallelizable. This is especially true in our field of electron cryomicroscopy, where the large number of individual images in almost every project leads naturally to easy separation. Here we have described and demonstrated a simple Perl-based distributed computation system called Peach, designed to distribute large numbers of independent jobs across the kinds of heterogeneous computer clusters commonly found in structural biology laboratories.

Here at Caltech, we have at present roughly twenty workstations scattered throughout the laboratory for interactive use. When fully loaded with jobs during normal weekdays, we found that Peach was able to use on average 69% of the capacity of these personal workstations, without disturbing interactive users. Whenever someone began using the console, even for undemanding applications such as word processing, Peach immediately suspended its jobs until the computer was once again idle. If a Peach application consumes all a client's memory, or worse causes major swapping, an annoying delay could be experienced as it is moved to the background. While we have not yet encountered this problem, we expect it would be similar to the delay caused by a complex, memory-intensive screen saver. The fact that Peach still took advantage of over two-thirds of the workstations' total potential is easily rationalized by recognizing that a regular "full-time" job accounts for only about one-fifth of the hours of a year, and further considering that the average researcher spends a great deal of time away from his/her desk even during workdays. In addition to the personal workstations, we also have some processors assembled as "compute clusters" with no monitor. Peach used these simultaneously with the personal workstations to 99% efficiency, demonstrating its ability to pool the power of such dedicated machines with the others in the laboratory.

By facilitating the use of all the available computer power, Peach has allowed us to finish projects that would otherwise have required expensive new hardware. In addition, Peach has accelerated our routine work and distributed resources more equitably by running each job on the fastest available processor, regardless of whose desk it is sitting on.

Peach is distinguished from other distributed systems by its simplicity and ease of use. There are only three user commands: one to submit jobs, one to monitor the status of jobs and processors, and one to kill jobs. A job is submitted simply by listing it, along with necessary flags and options, as arguments to *psubmit*. Peach is immediately compatible with any non-interactive command-line executable including scripts and, notably, commands in all the commonly used cryo-EM image processing packages. Installation is accomplished by running a single script which copies Peach onto a shared disk, launching the server and client daemons, and registering the users. No compilations or special libraries are required, and Peach will run on any Unix machine with a recent (less than five years old) version of Perl. Because Peach uses the modular client-server approach, it is robust to most common computer failures including loss of any of the processes, loss of any of the workstations, and delays in network communications. It remains sensitive, however, to failures of the shared disks, so choosing a reliable disk server is important.

Access to the Peach system is controlled by registration and passwords. To avoid interception, passwords are never sent in a clear text form. User registration also allows Peach to run jobs on computers without the need for user accounts on those machines, as long as the shared disk is mounted and the user has permission to read and write to the shared disk. We have not discovered any security loopholes thus far, and believe that the code's shortness and simplicity reduce vulnerability as compared to other existing packages.

We anticipate that Peach will be used on large clusters of computers. To assess its ability to serve such large clusters, we ran simulations where hundreds of job clients were launched. These demonstrated that up to a thousand computers can be handled well by a single job server through the adjustment of one parameter, the job client reporting interval. Thus Peach can handle even the largest modern clusters. Faster computers in the future will increase the capacity of the server, and configurations with multiple servers could be used to further extend the scale, if ever necessary.

One of the design goals was that Peach be immediately compatible with the hardware and software resources of typical structural biology laboratories. While Peach does work with any command-line Unix executable, the GUIs and command-line interpreters present in many packages would have to be adapted to take advantage of Peach's distributing potential. Among the most common packages used for cryo-EM-based single particle analysis are, for example, Spider, Imagic, and EMAN (Frank, Radermacher et al. 1996; van Heel, Harauz et al. 1996; Ludtke, Baldwin et al. 1999). Spider batch jobs, which are launched from the command line, could be distributed as a single job by Peach, which would help in a situation where multiple batch jobs were being submitted simultaneously within a laboratory. In particular, Peach would make it easy to send jobs away from the computer being used to submit the job, preventing slowdowns. Similarly, Imagic's batch accumulation mode assembles a c-shell script which could be distributed by Peach, as could EMAN script files or individual EMAN programs. The command-line interpreters and GUIs asociated with these packages, however, would have to be modified before the jobs they launched could be managed by Peach. Spider, Imagic, and EMAN already provide powerful built-in capacities to exploit homogeneous clusters. Peach's ability to distribute jobs across heterogeneous clusters should be viewed as complementary. The ideal system would efficiently access all resources (homogeneous and heterogeneous clusters) through all interfaces (command-line executables, command-line interpreters, and GUIs). As long as computational tasks did not require inter-process communication, but instead could be broken down into a large number of separate small processes, the principles we used to develop Peach could be used to achieve this. Command-line interpreters and GUIs would have to be modified to submit jobs to a Peach-like system, and Peach would have to be modified to parse large scripts defining entire image processing pipelines and launch jobs sequentially or in parallel, as appropriate. The Peach package is freely available at http://www.jensenlab.caltech.edu, or upon request to the authors.

2.6 Acknowledgements

We thank C. Iancu for her willingness to test and use Peach during development stages; P. Ober for the early development of ideas for distributed processing; and W. Tivol, S. Tivol, and D. Morris for reviewing the manuscript. This work was supported in part by NIH Grant PO1 GM66521 to GJJ, DOE grant DE-FG02-04ER63785 to GJJ, the

Beckman Institute at Caltech, and gifts from the Ralph M. Parsons Foundation, the Agouron Institute, and the Gordon and Betty Moore Foundation to Caltech.

2.7 References

Anderson, D. P., J. Cobb, et al. (2002). "SETI@home - An experiment in public-resource computing." Communications of the Acm **45**(11): 56−61.

Cowley, J. M. and A. F. Moodie (1957). "The Scattering of Electrons by Atoms and Crystals .1. a New Theoretical Approach." Acta Crystallographica **10**(10): 609−619.

Cowley, J. M. and A. F. Moodie (1957). "The Scattering of Electrons by Atoms and Crystals. I. A New Theoretical Approach." Acta Cryst. **10**: 609-619.

Fernandez, J. J., A. F. Lawrence, et al. (2002). "High-performance electron tomography of complex biological specimens." Journal of Structural Biology **138**(1−2): 6−20.

Frangakis, A. S. and F. Forster (2004). "Computational exploration of structural information from cryo-electron tomograms." Current Opinion in Structural Biology **14**(3): 325−331.

Frank, J. (2002). "Single-particle imaging of macromolecules by cryo-electron microscopy." Annual Review of Biophysics and Biomolecular Structure **31**: 303−319.

Frank, J., M. Radermacher, et al. (1996). "SPIDER and WEB: Processing and visualization of images in 3D electron microscopy and related fields." Journal of Structural Biology **116**(1): 190−199.

Glaeser, R. M. (1999). "Review: Electron crystallography: Present excitement, a nod to the past, anticipating the future." Journal of Structural Biology **128**(1): 3−14.

Heymann, J. B. (2001). "Bsoft: image and molecular processing in electron microscopy." J Struct Biol **133**(2-3): 156-69.

Lowe, J., D. Stock, et al. (1995). "Crystal structure of the 20S proteasome from the archaeon T. acidophilum at 3.4 A resolution." Science **268**(5210): 533-9.

Ludtke, S. J., P. R. Baldwin, et al. (1999). "EMAN: Semiautomated software for high-resolution single-particle reconstructions." Journal of Structural Biology **128**(1): 82−97.

Mouche, F., Y. Zhu, et al. (2003). "Automated three-dimensional reconstruction of keyhole limpet hemocyanin type 1." J Struct Biol **144**(3): 301-12.

Orlova, E. V. and H. R. Saibil (2004). "Structure determination of macromolecular assemblies by single-particle analysis of cryo-electron micrographs." Current Opinion in Structural Biology **14**(5): 584−590.

Sali, A., R. Glaeser, et al. (2003). "From words to literature in structural proteomics." Nature **422**(6928): 216−225.

Subramaniam, S. and J. L. S. Milne (2004). "Three-dimensional electron microscopy at molecular resolution." Annual Review of Biophysics and Biomolecular Structure **33**: 141−155.

Tannenbaum, T. and M. Litzkow (1995). "The Condor Distributed-Processing System." Dr Dobbs Journal **20**(2): 40−48.

van Heel, M., B. Gowen, et al. (2000). "Single-particle electron cryo-microscopy: towards atomic resolution." Quarterly Reviews of Biophysics **33**(4): 307−369.

van Heel, M., G. Harauz, et al. (1996). "A new generation of the IMAGIC image processing system." Journal of Structural Biology **116**(1): 17−24.

Walz, T. and N. Grigorieff (1998). "Electron crystallography of two-dimensional crystals of membrane proteins." Journal of Structural Biology **121**(2): 142−161.

2.8 Figures

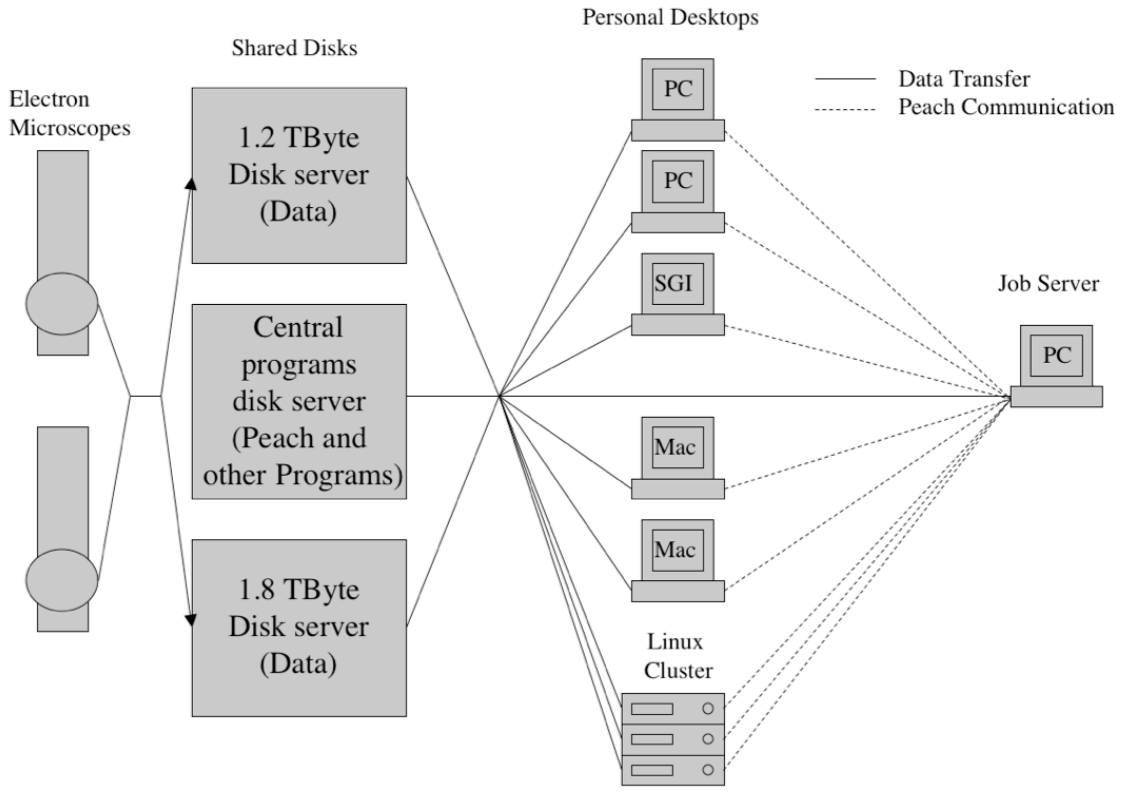

Figure 2-1. Schematic drawing of the setup and information flow in the testing of Peach. Image data was collected on two electron microscopes and transferred to two shared data disks. All the personal workstations located on desks throughout the laboratory and the several processors of a monitor-less compute cluster were configured to mount a central shared programs disk and the two data disks. Any particular workstation could host the job server. Users submitted jobs to Peach from their personal workstations. Information about each job was passed to the job server, which distributed jobs to idle workstations. Workstations retrieved job data from and wrote results to the shared disks. Solid lines represent job data transfer and dotted lines represent Peach network messages

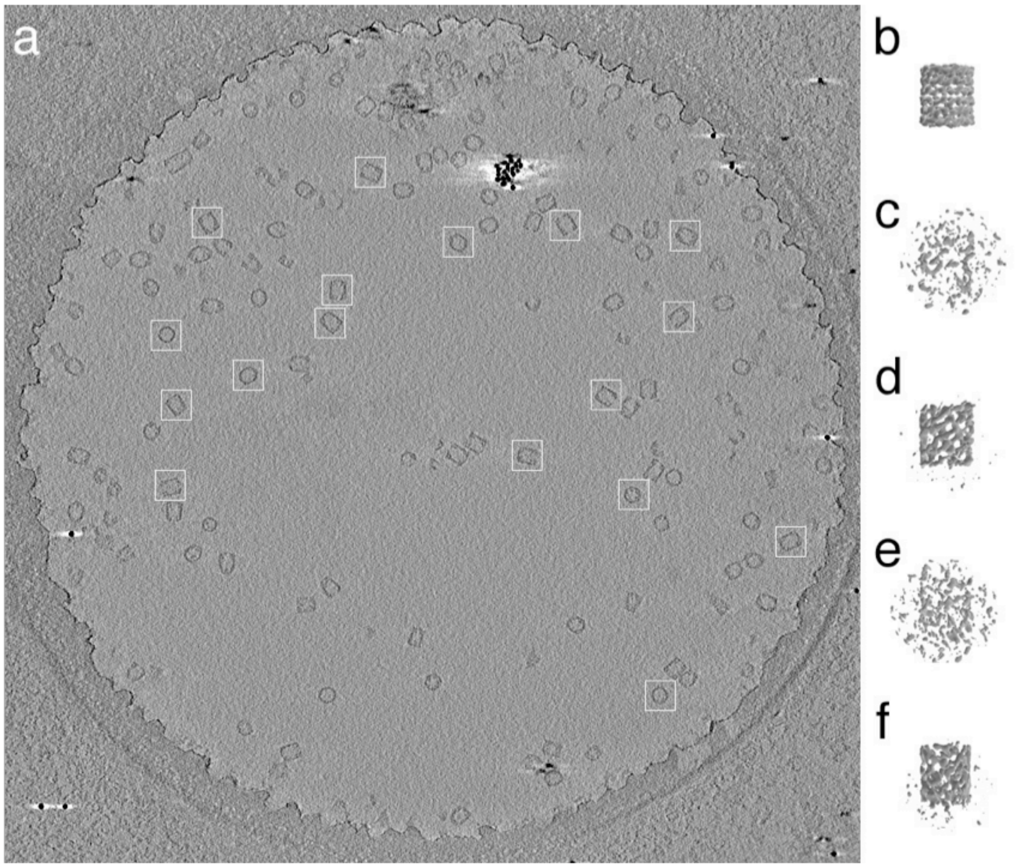

Figure 2-2. An example cryoEM image processing project made feasible by Peach. Peach managed extensive calculations comparing electron tomograms recorded with different electron doses and different sample temperatures. The sample was the 35 nm long, barrel-shaped protein complex hemocyanin, purified and suspended within a thin film of vitreous ice across circular holes in a supporting carbon film. (a) A single section through a tomogram, where several individual hemocyanin molecules are marked with square boxes. The small black dots are colloidal gold fiducial markers. (b) 12 Å structure of hemocyanin (Mouche, Zhu et al. 2003) used as template. (c–f) Representative three-dimensional reconstructions of individual hemocyanin molecules, extracted from tomograms recorded at liquid nitrogen (c,d) or helium (e,f) temperature, with doses of 10 (c,e) or 120 (d,f) electrons/Å^2, oriented using the template in (b)

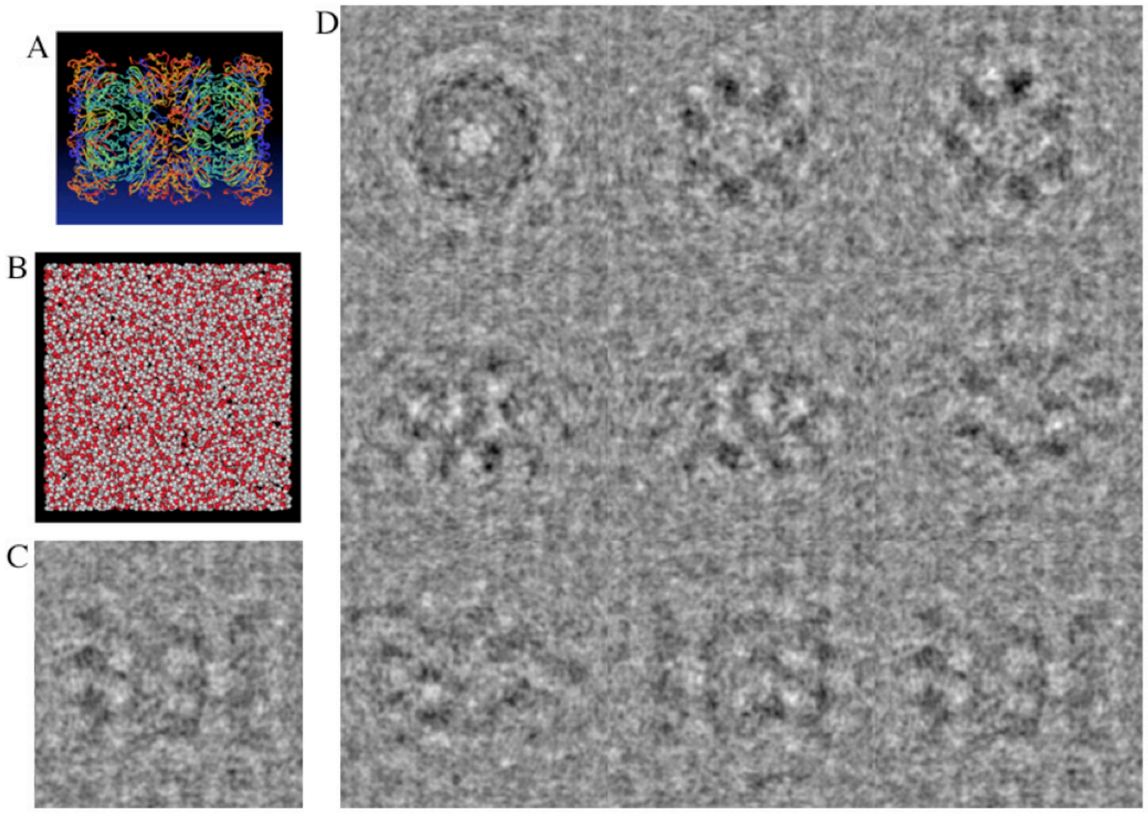

Figure 2-3. An example image simulation project managed by Peach. Peach was used to simulate thousands of cryo-EM images of a water-embedded protein from different points of view and under different imaging conditions using a multi-slice algorithm (Cowley and Moodie 1957). (a) A ribbon diagram of the test protein, the 20S proteasome (Lowe, Stock et al. 1995). (b) Block of water used to embed the test protein. (c) Simulated cryo-EM image of the 20S protein embedded in water from the same point of view as in (a). (d) Montage of nine other simulated images, showing the 20S protein from various points of view

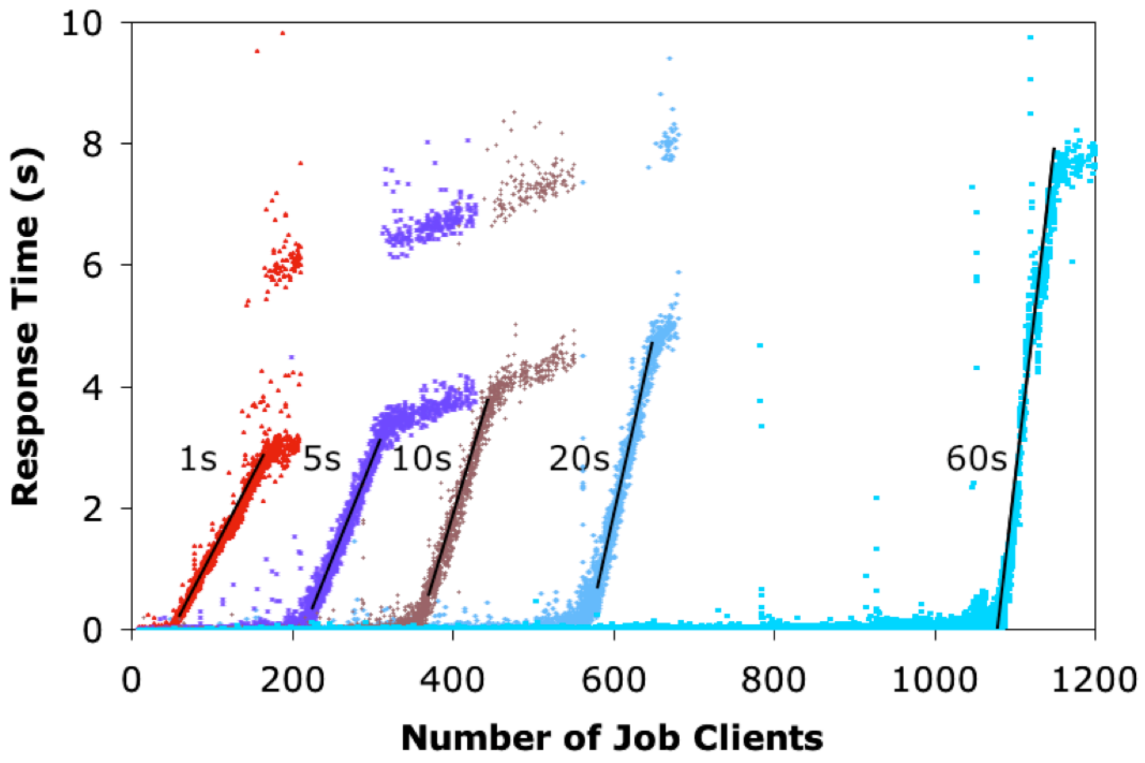

Figure 2-4. Scalability. The ability of Peach to manage large numbers of computers was tested by adding job clients to the system incrementally while measuring the delay between job client reports and the job server's response. The results from five separate tests are shown, in which the job client reporting interval (the time each job client waited before sending its next report) was set to 1, 5, 10, 20, and 60 s. Each graph shows three distinct regions. In the first region, the job server is unsaturated and responds to job clients immediately. As additional job clients are added, the server eventually becomes saturated, socket queues begin to fill, and the response time increases linearly. Finally, socket queues also become saturated and some connections are refused, generating erratic response times. For these tests, the job server was a 2.4 GHz IBM PC with 1.5 gigabytes of memory running Redhat Linux