A Data Management Architecture for Computational Biology

Yu Ma, Randall Bramley, Sun Kim
Indiana University
Bloomington, Indiana
{yuma,bramley,sunkim2}@indiana.edu

Abstract

With increasing availability of high-performance computer clusters, a key bottleneck in computational biology research is the handling of metadata: information about computational jobs, locations of files, staging input and output data where it is needed, and application-specific data allowing searching and querying the results of computations. By examining the needs of PLATCOM, a gene similarity platform, general requirements are derived for a data management architecture, from which particular data management systems can be quickly composed. The prototype architecture is described and its application to genome comparative analysis in PLATCOM is shown in detail.

1. Introduction

Computational biology applications commonly need large scale data management of input/output files, access to shared and private databases, and the managment of data related to running large numbers of jobs on parallel computers at remote sites, typically through a batch manager and scheduler. The data may require curation unrelated to the actual bioinformatics computation, such as transfer to and from archival tape storage. The scientific goals of computational biology (e.g., gene expression, evolutionary tree optimization, data mining) and methods are varied and quickly evolving, so a single data managment system is unlikely to be optimal or even suitable for all laboratories, or even a single laboratory as its purview evolves. Instead, the ability to rapidly create customized data management systems is a potentially enabling technology for bioinformatics.

An example of this need is PLATCOM (Platform for Computational Comparative Genomics) [13], an integrated computational biology system for the comparative analysis of multiple genomes. One of the internal databases that PLATCOM incorporates is derived from GenBank [12] by performing pairwise comparisons of protein-to-protein and whole genome-to-whole genome sequences using FASTA [20] and BLASTZ [22] respectively, which entails over 48,000 similarity computations initially, and several hundreds more every couple of months afterwards as the GenBank grows. Each new addition to GeneBank leads to q additional computations, where q is the current number of sequences in GeneBank. These computations need to be automatically triggered and managed without requiring direct intervention by a researcher.

The large number of computation jobs in PLATCOM has presented several data management challenges such as input/output file staging, statistical metadata management, resource acquisition, and fault-tolerant job submission. Many scientific applications face similar problems, but often vary on their detailed requirements. Rather than building a monolithic system that attempts to handle all aspects of data management, a different approach has been developed: a component-like architecture consisting of utilities such that each does one simple task well, and together they can be rapidly and easily assembled to create customized data management systems.

Common scientific data management needs have been coarsely classified, and a set of functionally orthogonal components identified, each dedicated to solving one simple problem. Since different applications often require or prefer different computation environments, and some existing software is already mature and widely-used for certain tasks, a goal was to reuse existing tools whenever possible. A layered style architecture [23] is thus presented that supports increasing levels of abstraction, enhancement and reuse. Such an architecture is flexible and easy to adapt to different platforms and software tools used at lower level, while at the same time providing a uniform interface for developing higher level application-specific data management systems. This provides a general solution for diverse scientific data management problems using a well-known computer science dictum[18]:

> Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.

This prototype component architecture for data management (Obsidian) is being used for computational biology and other scientific applications. This paper first closely studies PLATCOM data management requirements and why some popular existing systems are not desired in this case. The functional component set of Obisidian is then derived, generalized, and finally applied to PLATCOM.

2. Data Management Requirements in PLATCOM

PLATCOM [13] provides an integrated comparative genome analysis environment that facilitates the design of experimental protocols. It consists of a set of internal databases and a suite of genomic analysis tools for plotting, visualization, gene fusion event detection, metabolic pathway analysis and gene clustering.

Most internal databases such as GenBank [12], Swiss-Prot [10], COG [24], and KEGG [2] are downloaded from public domains. In order to increase the performance of real-time multiple genome comparison computation, PLATCOM also pre-computes and maintains a Pairwise Comparison Database (PCDB). Initially, PCDB consists of 97,034 pairwise comparison entries for both protein sequence files (.faa) and whole genome sequence files (.fna) of 312 replicons, which currently entails 48,516 FASTA computations to generate unduplicated protein-to-protein pairwise comparison matches.

FASTA [20] programs provide sequence similarity searching against nucleotide and protein databases. The fasta34 series can run both as threaded parallel and distributed memory parallel using PVM and MPI. Two input files define the two sequences to be compared, and an output file contains a detailed report including a histogram, list of hits, and alignments.

To improve storage and performance, PLATCOM processes the FASTA output further to extract only useful information (id, z-score, e-value, SW score and match region) for both query and matching sequences, and stores the resulting ASCII file in PCDB. This parsing procedure is tightly coupled with each FASTA computation performed. PCDB follows the same UNIX filesystem directory structure as that of GenBank, and creates a subdirectory under the query protein (as opposed to matching protein) to store the parsed FASTA results.

The initialization and continuous evolvement of PCDB is a tedious task without help from a comprehensive data management system, which can automatically stage input and output files, interface with batch managers on remote supercomputer facilities to carry out FASTA computations and result parsings, securely transfer files between the remote and the local site, and possibly create an archival copy on backup tape. This data management system is also needed to keep track of historical information (i.e. metadata) about job submission, completion status, and computation performance to facilitate failed job discovery and future statistical analysis.

For each computational job, the metadata that PLATCOM needs include sequence file locations, participating protein IDs, hardware and operating system configurations, who submitted the job using which version of FASTA with what parameter settings, when the jobs was submitted, how long did the computation and result parsing take, what was the CPU and memory usage, and the exit statuses of the job. Multiple exit statuses occur from the FASTA job as a Unix process, the exit status returned by the batch manager, that of a harness script managing the job submission to the batch manager, and even the scientist's satisfaction with the results.

These metadata are both general and specific. General in the sense that any one of them could be of concern to some other applications, while specific in the sense that PLATCOM cares about only this particular combination of them. The data management system to be developed should efficiently manage just this set of metadata and not require the end user to deal with irrelevant categories.

3. Related Work

Until recently *ad hoc* data management in computational biology and most other scientific computation was adequate. Running 48,000 parallel jobs could take years to complete, providing plenty of time for scientists to stage input files, move output results to longer-term storage, and relaunch failed jobs by hand. Data would typically start on the computational platform and output results would remain there until analyzed. The increasing availability of fast parallel platforms and cheaper local hard drive storage now means that the rate of generation of scientific metadata from computation exceeds the capabilities of approaches that require human intervention in the minutia of data management.

Distributed filesystems such as AFS and NFS aid in hiding the complexity introduced by the use of distributed systems, by making remote files appear to be in a local directory. This requires coordination between administrative domains for user identification and authentication, and can cause computational job failures if a remote input file is not immediately available because of network problems. In general a distributed filesystem does not coordinate data and workflow, and is at best a partial metadata solution.

The SDSC Storage Resource Broker (SRB) [21] can provide distributed clients with a uniform interface to access heterogeneous underlying data storage resources such as databases, filesystems, and tape storage. Meta-

Data Catalog (MCAT) [4] is a metadata repository mainly in support of resource discovery for SRB. MCAT categorizes general metadata according to five primitive entities, and distinguishes the notion between system-level metadata and application-level metadata, i.e. information not generalizable as primitive-related metadata are application-level metadata. MCAT architecture design allows the creation of arbitrary application-level metadata, although the corresponding implementation level support is limited.

MCAT has introduced a comprehensive metadata management architecture and some excellent concepts for metadata categorization. However, it provides neither a solution adequate for PLATCOM or a panacea for scientific metadata management in general. The system-level metadata schema MCAT chooses best represents the information that SRB requires, and the application-level metadata support is limited to string text which is not searchable via relational queries. From the view of PLATCOM data management needs, most metadata in that schema is redundant, while at the same time, the schema is not sufficient to represent and query the execution and performance metadata needed. Other applications face similar situations when using MCAT, which is an inherent problem of generalizing a universal metadata schema in practice.

As with any single integrated system, MCAT also has several implementation level limitations, the biggest one of which for PLATCOM is the complexity of system installation and maintenance. MCAT is tightly coupled with SRB so that one can not be easily installed and work correctly without interacting with the other. Experience has shown that it takes significant administrative effort to configure and build an MCAT-enabled SRB system as a long-term reliable service.

The Condor [11] workload management system from the University of Wisconsin-Madison can seamlessly combine distributed computational power into one resource, and transparently migrate a job from one machine to another without requiring a shared filesystem across them. It can succeed with many job management tasks, especially effectively harnessing wasted CPU power from otherwise idle desktop workstations. However, it targets job management instead of the metadata and workflow management as needed by PLATCOM, and similarly to MCAT, Condor requires non-trivial administrative effort and privileges.

Many supercomputer facilities or workstation clusters use batch managers such as PBS [7] to enforce resource allocation policies among users. This is often sufficient for resource acquisition of most computional biology applications including PLATCOM. However, neither Condor or popular scientific workflow management systems like KEPLER [9] and Triana [17] address one particular need of PLATCOM, which is to interface with an existing batch manager like PBS, and automatically submit a large number of simple workflow jobs.

Several other major on-going data management efforts are either grid-oriented and more suitable for large collaborations like the GeneGrid [15], or more specialized for a particular problem or environment that is not applicable to PLATCOM. Scientific Annotation Middleware (SAM) [19] provides a set of services to support the creation and semantic translation of annotation metadata; Chimera Virtual Data System [14] aims at representing data derivation procedures and enabling on-demand data (re)generation; Laboratory Information Management Systems (LIMS) [25] are widely used in pharmaceutical and life science applications, and often come as off-shelf commercial products, mostly PC-based for Windows operating systems.

4. Obsidian Data Management Architecture

PLATCOM data management requirements lead to the following orthogonal components needed to build a customized solution: a secure file transfer mechanism, a lightweight relational database backend, a metadata schema about FASTA job submission, execution and performance, a mechanism to keep track of input/output file locations, and finally a fault-tolerant automatic job (re)submission method.

Mature products already exist for secure file transfer, e.g. SCP and GRIDFTP [1]. Relational database backend also has many possible choices. With the preference for open source and both Windows and Linux implementations, MYSQL [5] and POSTGRESQL [8] are available. A relational schema for job submission, execution and performance metadata is easily defined in SQL, as well as a relational schema for keeping track of file locations in the database. Since PLATCOM computation jobs only differ in their input protein sequence files, automatically submitting thousands of such jobs to a local batch manager and scheduler can be implemented easily in many scripting languages.

However, the challenge lies in tying all the pieces together in a distributed environment and making the job submission procedure fault-tolerant. When a job fails and no output file is produced as expected, the reason should be readily identifiable: is it at the software level such as wrong parameter settings and corrupted input files, or at the hardware level such as a crash of a compute node? The locations of the original pair of protein sequence files also need to be identified to resubmit the job. These tasks are not hard to do manually with only a few dozens of jobs to manage, but become impractical without a good data management support when the number goes to thousands.

A more important question is how to build a portable and reusable solution. While another scientific application is unlikely to have exactly the same data management requirements as PLATCOM, many scientific applications are facing similar kinds of data management challenges. They may require a different metadata schema, work with a different operating system, or use a different relational database. But they all need to keep track of their data object locations, store, update and query their own metadata, or securely transfer data objects between local and remote sites, between disk and tape storage, etc.

Obsidian is a component-like layered architecture that addresses this issue. Obsidian builds a general scientific data management solution not as a generic system for all applications, but instead as a toolkit for each application to compose its own specialized system. This approach is particularly suitable to meet data management needs from individual scientists or moderate sized collaborations, where an efficient and light-weight solution is typically preferred. Common scientific data management involves the following three major aspects about data objects:

- Physical accessibility: i.e. keep track of physical locations of data objects and securely transfer them as desired for access or storage;
- Logical relationship: i.e. categorize data objects into different collections according to pre-defined logical relations to better characterize relationships among data objects, and therefore their corresponding application components;
- Information representation: i.e. define, store, and query information of interest related to, or represented by data objects for statistical or analytical studies.

Unique identifiers, either universally or within an application domain, are often needed to cross-reference data objects and logical collections with other information. From these common data management needs, the following independent functional components to facilitate building specialized data management systems have been derived:

- **Unique ID Generator** that generates either universally or application-specific unique identifiers;
- **Data Object Accessor** that provides I/O operations functionally similar to cp, mv, rm, zip and tar on distributed data objects across both disk and tape storage;
- **Physical Location Tracker** that defines a relational schema on application-independent metadata about data objects (e.g. type, size, location), and provides interfaces to archive, query, and update these metadata, especially the physical data object locations;
- **Logical Collection Manager** that provides a mechanism to create, update, query and delete identifier collections in a relational database based on arbitrary user-defined relationships;
- **Annotation Manager** that manages notes and annotations about identifiers as text blobs, and also provides corresponding create, update, retrieve and delete interfaces to the underlying database;
- **MDB Handler** similar to ODBC (Open Database Connectivity) [6], and provides a simple programming interface to allow SQL query and update on application-specific metadata databases, assuming the user knows the underlying relational schema.

The components defined above are orthogonal to each other and can be combined in different ways to meet different application needs. Note that no generalized metadata schema is enforced, because it is often more straightforward and efficient to first define application-specific metadata relations in the database directly. This is a one-time effort for each application, and afterwards the MDB Handler is used to access and manage the metadata programmatically.

Obsidian consists of three layers. At the bottom layer are basic component units of various existing, widely-used, fundamental tools, ranging from low-level filesystem calls such as CP and MV, to full-fledged mature software packages like GRIDFTP and MYSQL. The middle layer includes the set of functionally orthogonal components as defined above, implemented using bottom layer component units. At the top are specialized data management systems (DMS) built with some or all of the middle layer components. One middle layer component may have multiple implementations using different bottom layer units, all of which provide the same functionality and interface to the top layer. For example, the copy function of Data Object Accessor component can be implemented using UNIX CP, SCP, GRIDFTP, or HSI to duplicate an existing data object at a different location. Middle layer components may also be implemented in high level scripting languages such as Perl, which are platform independent. This organization has allowed easy plug in and out bottom layer units to meet different application requirements, while not affecting the programming interface provided by the middle layer components for building a specialized top layer data management system. With the given maximum freedom of choosing bottom layer supporting units a customized solution can be built with minimal administrative privileges, sometimes strictly in user space. Since middle layer components are independent with each other, Obsidian is naturally extensible as more functional middle layer components are identified.

Figure 1 depicts the overall Obsidian architecture as described above. Highlighted pieces are those used for creating the PLATCOM solution discussed in Section 5.

5. PLATCOM Job Management Solution

Currently all PLATCOM computation jobs are carried out on a remote IA32-based Linux cluster called AVIDD

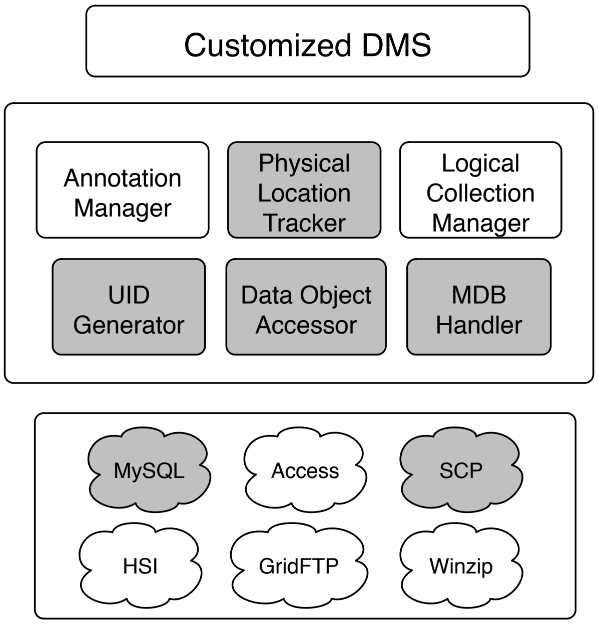

Figure 1. Obsidian Components

at Indiana University Bloomington campus. AVIDD employs PBS Pro [7] and Maui [3] to manage computing resources and schedule jobs. All input sequence files and parsed FASTA results are stored on a local workstation called platcom, outside the AVIDD administrative domain.

Based on the Obsidian architecuture, a comprehensive job management solution (JMS) has been implemented to coordinate data and workflows between AVIDD and platcom. The JMS uses MYSQL as the backend metadata database (MDB), and SCP as the secure file transfer mechanism. By composing highlighted Obsidian middle layer components in Figure 1, all of the functionality described below has been implemented with fewer than 300 lines of Perl code.

JMS takes as input the complete list of current protein sequence files. Figure 2 demonstrates the job management procedure in detail.

1. For each given protein pair to be compared, JMS generates a corresponding PBS job submission script that archives submission metadata, invokes parallel FASTA, and summarizes comparison results.

2. It then transfers corresponding input files and the submission script generated from platcom to AVIDD.

3. The job is in turn submitted to the PBS queue, waiting to be scheduled for execution when resources become available.

4. Upon completion, the parsed result file is transfered back from AVIDD to a designated location on platcom.

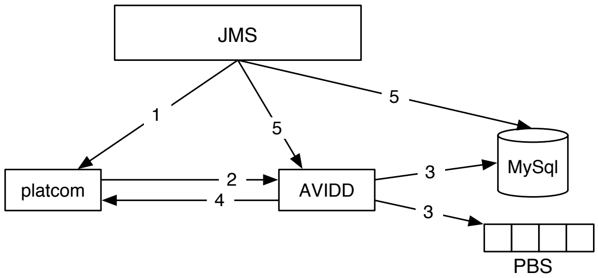

Figure 2. Job Management Workflow for Platcom

5. Periodically, JMS parses execution status reports generated by PBS and archives execution and performance metadata in MDB.

Tables 1–3 define MYSQL database tables describing the relational schema for job submission, execution and performance metadata in PLATCOM. Job submission meta-

Executions	
EID	Unique Execution ID
genome1	Unique input Genome ID
genome2	Unique input Genome ID
host	Computing host name
sysinfo	System hardware configuration
command	Complete computation command
timestamp	Job submission time
contact	User contact information
PBS_JOBID	Unique PBS Job ID
dest	Output file destination

Table 1. Job Execution Metadata

PBSJobReports	
EID	Unique Execution ID
exitstat	Computation exit status
cpupercent	CPU usage percentage
cputime	CPU time used (sec.)
mem	Memory used (kb.)
ncpus	Number of CPUs used
vmem	Virtual memory used (kb.)
walltime	Wall-clock time used (sec.)

Table 2. Batch Manager Metadata

Genomes	
GID	Unique Genome ID
name	Genome name
genbankTag	Corresponding GenBank Tag
loc	Absolute file location

Table 3. File Metadata

data in Table 1 is associated with job execution and performance metadata in Table 2 through the PBS_JOBID field. Since Table 1 is populated before the computation, while Table 2 is populated after, this layout allows updating one table without having to query another.

When a job execution failure is detected either from its exit status or abnormal hardware behavior, the above metadata schema allows quickly finding out the job submission metadata, particularly physical locations of corresponding input sequence files, so that the job failure cause can be analyzed and the job relaunched accordingly.

When running parallel jobs on a shared cluster like AVIDD, the trade-off between execution speedups by requesting more processors and extended the wait time in queue to get those processors must be balanced. To find the optimal number of processors to requst for each parallel FASTA job, a sample set of 400 randomly selected jobs was tested before going into the production mode. Those jobs were further equally divided into 5 groups, each group of jobs requesting 2, 4, 8, 16, and 32 processors for execution respectively. Figure 3 is the speedup plot based on the average execution time for each group. As the data clearly shows, four processors gives the optimal performance and actually results in superlinear speedup. So each parallel FASTA job requests four processors in the corresponding PBS script.

AVIDD enforces the usage policy of at most 64 running or idle jobs per user, to maintain system load balance. The resource availability rate varies greatly from time to time, the JMS incorporates an automatic monitoring mechanism which checks the queue status every 30 seconds and submits more jobs whenever the number of queued jobs falls below the maximum threshold. This mechanism has greatly enhanced the resource utilization on such a busy system as AVIDD.

In the production mode, it took 33 days for all 48,516 jobs to complete correctly, averaging 1470 jobs per day. During this time, AVIDD experienced several kinds of system failures, including Maui hanging, Myrinet card failure, failed NFS mounts and degraded GPFS (General Parallel Filesystem) services. Several dozens of jobs failed either directly because of these system failures, or because the execution time exceeded the expected maximum and the job was killed by PBS. JMS was able to recover from all these failures and relaunched failed jobs successfully.

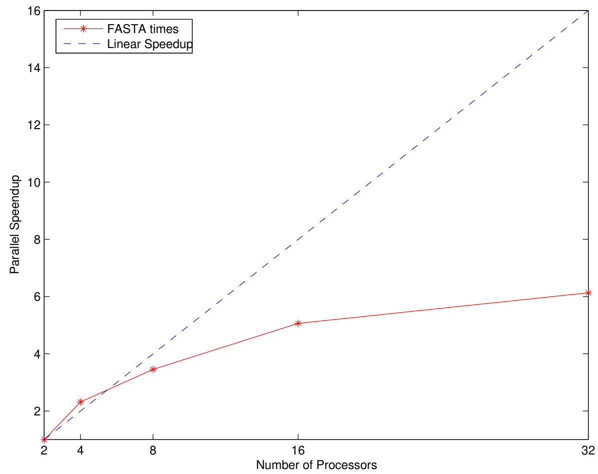

Figure 3. Parallel Speedups for FASTA

6. Status and Future Work

The Obsidian architecture's middle layer components have been implemented with several programming language bindings including Perl, C/C++ and Java. Besides PLATCOM, Obsidian is being used to build customized data management systems for a wide range of scientific applications such as x-ray diffractometry, radiation therapy health records, and automated photometry in astronomy. The applicability of the Obsidian architecute will be further studied, and possibly the orthogonal middle layer component set refined as new needs are encountered. In a near future, the interfaces of these components will be defined in Scientific Interface Definition Language (SIDL) [16].

Acknowledgments

This work is supported in part by National Science Foundation Grants ANI-0330568, EIA-0202048, and CDA-0116050.

References

[1] The GridFTP protocol and software. `http://www.globus.org/datagrid/gridftp.html`. The Globus Alliance.

[2] KEGG: Kyoto encyclopedia of genes and genomes. `http://www.genome.jp/kegg/`. Kanehisa Laboratory.

[3] Maui cluster scheduler. `http://www.clusterresources.com/products/maui/`. Cluster Resources, Inc.

[4] MCAT - a meta information catalog. http://www.npaci.edu/DICE/SRB/mcat.html. San Diego Supercomputer Center, NPACI Data Intensive Computing Environment.

[5] MySQL. http://www.mysql.com/. MySQL Inc.

[6] *The ODBC Solution, Open Database Connectivity in Distributed Environments*.

[7] PBS Pro. http://www.pbspro.com/. Altair Grid Technologies.

[8] PostgreSQL. http://www.postgresql.org/. PostgreSQL Global Development Group.

[9] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *16th International Conference on Scientific and Statistical Database Management (SSDBM)*, Santorini Island, Greece, June 2004. See also http://kepler.ecoinformatics.org/.

[10] A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger. Swiss-Prot: juggling between evolution and stability. *Briefings in Bioinformatics*, 5(1):39–55, March 2004. See also http://www.ebi.ac.uk/swissprot/.

[11] J. Basney and M. Livny. Deploying a high throughput computing cluster. In R. Buyya, editor, *High Performance Cluster Computing, Volume 1: Architectures and Systems*, chapter 5, pages 116–134. Prentice Hall PTR, 1999. See also http://www.cs.wisc.edu/condor/.

[12] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank: update. *Nucleic Acids Research*, 32(Database issue):D23–D26, 2004. See also http://www.ncbi.nlm.nih.gov/Genbank/.

[13] K. Choi, Y. Ma, J.-H. Choi, and S. Kim. PLATCOM : A platform for computational comparative genomics. *Bioinformatics*, Accepted. See also http://platcom.informatics.indiana.edu/platcom/.

[14] I. Foster, J. Vckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *14th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2002. See also http://www.griphyn.org/chimera/.

[15] N. Kelly, P. Jithesh, D. R. Simpson, P. Donachy, T. J. Harmer, R. Perrott, J. Johnston, P. Kerr, M. McCurley, and S. McKee. Bioinformatics data and the grid: The GeneGrid data manager. In *UK e-Science All Hands Meeting 2004 (AHM04)*, September 2004. See also http://www.qub.ac.uk/escience/projects/genegrid/.

[16] S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. Divorcing language dependencies from a scientific software library. In *10th SIAM Conference on Parallel Processing*, Portsmouth, VA, March 12-14 2001. LLNL document UCRL-JC-140349. See also http://www.llnl.gov/CASC/components/babel.html.

[17] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana: A graphical web service composition and execution toolkit. In *IEEE International Conference on Web Services (ICWS'04)*, San Diego, CA, June 06 - 09 2004.

[18] M. D. McIlroy, E. N. Pinson, and B. A. Tague. Unix time-sharing system forward. *The Bell System Technical Journal*, 57(6, part 2):1902, 1978.

[19] J. D. Myers, A. Chappell, M. Elder, A. Geist, and J. Schwidder. Re-integrating the research record. *COMPUTING IN SCIENCE & ENGINEERING*, 5(3):44–50, MAY/JUNE 2003. See also http://www.scidac.org/SAM/.

[20] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proceedings of the National Academy of Sciences USA*, volume 85, pages 2444–2448, April 1988. See also http://www.ebi.ac.uk/fasta33/.

[21] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage resource broker - managing distributed data in a grid. *Computer Society of India Journal, Special Issue on SAN*, 33(4):42–54, October 2003. See also http://www.npaci.edu/DICE/SRB/.

[22] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13(1):103–107, January 2003. See also http://bio.cse.psu.edu/.

[23] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Publishing, 1996. 242 pages.

[24] R. Tatusov, E. Koonin, and D. Lipman. A genomic perspective on protein families. *Science*, 278(5338):631–637, October 24 1997. See also http://www.ncbi.nlm.nih.gov/COG/.

[25] C. G. Thurston. LIMS/instrument integration computing architecture for improved automation and flexibility. *American Laboratory*, September 2004. See also http://www.thermo.com/informatics.