CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2011; **23**:2305–2314
Published online 27 June 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1782

SPECIAL ISSUE PAPER

Biocompute 2.0: an improved collaborative workspace for data intensive bio-science

Rory Carmichael [1,*,†], Patrick Braga-Henebry [2], Douglas Thain [3] and Scott Emrich [3]

[1] *Bioinformatics Core Facility, University of Notre Dame, Notre Dame, IN 46556,USA*
[2] *IMC Financial Markets, 233 South Wacker Drive #4300, Chicago, IL 60606,USA*
[3] *Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556,USA*

SUMMARY

The explosion of data in the biological community requires scalable and flexible portals for bioinformatics. To help address this need, we proposed characteristics needed for rigorous, reproducible, and collaborative resources for data-intensive science. Implementing a system with these characteristics exposed challenges in user interface, data distribution, and workflow description/execution. We describe ongoing responses to these and other challenges. Our Data-Action-Queue design pattern addresses user interface and system organization concepts. A dynamic data distribution mechanism lays the foundation for the management of persistent datasets. Makeflow facilitates the simple description and execution of complex multi-part jobs and forms the kernel of a module system powering diverse bioinformatics applications. Our improved web portal, Biocompute 2.0, has been in production use since the summer of 2010. Through it and its predecessor, we have provided over 56 years of CPU time through its five modules—BLAST, SSAHA, SHRIMP, BWA, and SNPEXP—to research groups at three universities. In this paper, we describe the goals and interface to the system, its architecture and performance, and the insights gained in its development. Copyright © 2011 John Wiley & Sons, Ltd.

Received 1 October 2010; Revised 26 March 2011; Accepted 17 April 2011

KEY WORDS: bioinformatics; web portal; makeflow; interface design

1. INTRODUCTION

The field of biology is becoming increasingly reliant on computation. Data collection machines and methods continually decline in cost and increase in throughput, generating a profusion of data. This explosion of data has grown the collaborative field of bioinformatics, which has in turn generated many powerful tools for the exploration of large biological datasets. However, despite the convenient parallelism and demanding computational requirements present in many of these applications, there continues to be a dearth of easily deployable parallel bioinformatics tools. Although many popular resources for bioinformatic analysis exist, data acquisition has rapidly outpaced their analysis and curation capabilities.

We propose a more collaborative strategy for data intensive scientific computing. Our key goals are to provide collaborative, rigorous, and repeatable analytic results. To achieve these goals, we proposed a system in which user data are primary concerns, computational resources can be shared and managed, data and computational workflows are parallel, distributed, and replicable, and parameters

*Correspondence to: Rory Carmichael, 351 Fitzpatrick Hall, Notre Dame, IN 46556,USA.
†E-mail: rcarmich@nd.edu

Contract/grant sponsor: NSF; contract/grant number: CNS0643229

of analysis are persistently maintained [1]. By allowing scientific communities to pool and curate their data and resources in this way, we hope to increase the scalability of scientific endeavors beyond the scope of monolithic central resources.

To support this collaborative structure, we suggested the Data-Action-Queue interface [1]. In this paper, we refine a proof of concept implementation of this interface for the Bicompute 2.0 portal. Biocompute serves community needs by providing an environment where datasets can be readily shared and analyzed, results are automatically documented and easily reproducible, and new tools can be readily integrated. It runs on top of Chirp and Condor to facilitate distributed storage and computation on shared resources. As of the writing of this paper, Biocompute has provided over 56 years of CPU time through its five modules—BLAST, SSAHA, SHRIMP, BWA, and SNPEXP—to researchers at three universities [2, 14–16].

In this paper, we expand earlier work wherein we described the goals and interface to the system, its architecture and performance, and the insights gained in its development [1]. Dataset distribution, effective description and execution of workflows, and user interface all proved to be challenging in this context. In previous work, we provided a detailed description and evaluation of our data distribution solutions. Further, we expand our description of the Data-Action-Queue interface model and focus our attention on the module system underlying recent expansions in Biocompute's capabilities.

2. SYSTEM GOALS

Bioinformatics portals range from broad-base resources, such as NCBI [2], to more specific community level resources, such as VectorBase [3], down to organism or even dataset level web portals. Although these portals all rely, at least in part, on the scientific communities they support for data and analysis, they share the characteristics of centralized computation and curation. Additionally, many existing portals suffer from imperfect data transparency or job parameter storage, reducing the rigor and reproducibility of the results generated. As increasing number of organisms are sequenced, smaller and less well-funded biological communities are acquiring and attempting to analyze their data. These communities rarely have the resources to support the development of specialized community web portals, and find portals such as NCBI insufficient for their computation, customization, or collaboration needs.

It seems natural, then, to turn to a more rigorous, reproducible, and collaborative way to do data-intensive science. We believe the following characteristics to be vital to these goals.

1. User data must integrate with the system just as well as curator provided data.
2. Data management should be simple for owners as well as site administrators.
3. Sharing of data and results should be straightforward.
4. Persistent records of job parameters and metadata need to be kept.
5. Jobs should be easily resubmittable in order to reproduce results.
6. System resources should be shared fairly, productively, and transparently.
7. The resources providing computation and data storage must be scalable and shareable.

A system exhibiting these characteristics should permit a user community to develop and improve a shared resource that is capable of meeting their computational needs and contains the data they require. Further, it will allow users to maintain a clear, traceable, record of the precise sources of datasets and results.

3. SYSTEM OVERVIEW

3.1. Components

Biocompute is arranged into three primary components. A single server hosts the website, submits batch jobs, and stores data. A relational database stores metadata for the system such as user data, job status, runtime and disk usage statistics. Each dataset is stored in a distributed cache [4] over a cluster of 32 machines that have been integrated into our campus grid computing environment [5]. These machines serve as a primary runtime environment for batch jobs and are supplemented by an

DOI: 10.1002/cpe

extended set of machines running the same distributed caching system and advertising availability for Biocompute jobs using the Condor classad system [6].

3.2. *Interface: Data-Action-Queue*

Having the described functionality is insufficient if users cannot effectively use the resource. To provide the requisite interface, we employ the Data-Action-Queue (DAQ) interface design pattern. Like Model-View-Controller, this suggests a useful structure for organizing a program. However, DAQ describes an interface, rather than an implementation.

The DAQ design pattern rests on the idea that users of a scientific computing web portal will be interested in three things: their data, the tools by which they can analyze that data, and the record of previous and ongoing analyses. This also suggests a modular design for the implementing system. If tool developers need only specify the interface for the initial execution of their tool, it greatly simplifies the addition of new actions to the system. The queue view documents job parameters and meta information, and permits users to drill down to a single job in order to resubmit it or retrieve its results. Because the queue also shows the ongoing work in the system, it gives users a simple way to observe the current level of resource contention.

4. SYSTEM INTERFACE IMPLEMENTATION

In accordance with the DAQ model, users have three separate views of Biocompute: a filesystem-like interface to their data, a set of dialogues permitting users to utilize actions provided by the system, and a queue storing the status, inputs, and outputs of past and current jobs. Figure 1 shows our implementation of this model, and the following sections describe the interaction and sharing characteristics of its components.

Recently, we have moved to a more mature implementation of the Data-Action-Queue paradigm relying heavily on the insights of web design. It features an extremely flat interface—the deepest functionality is two clicks from the front page—and a REST-ful strategy that greatly facilitates natural collaboration by permitting users to share urls. The site also provides users with periodically updated statistics regarding disk and CPU usage and warnings when needed.

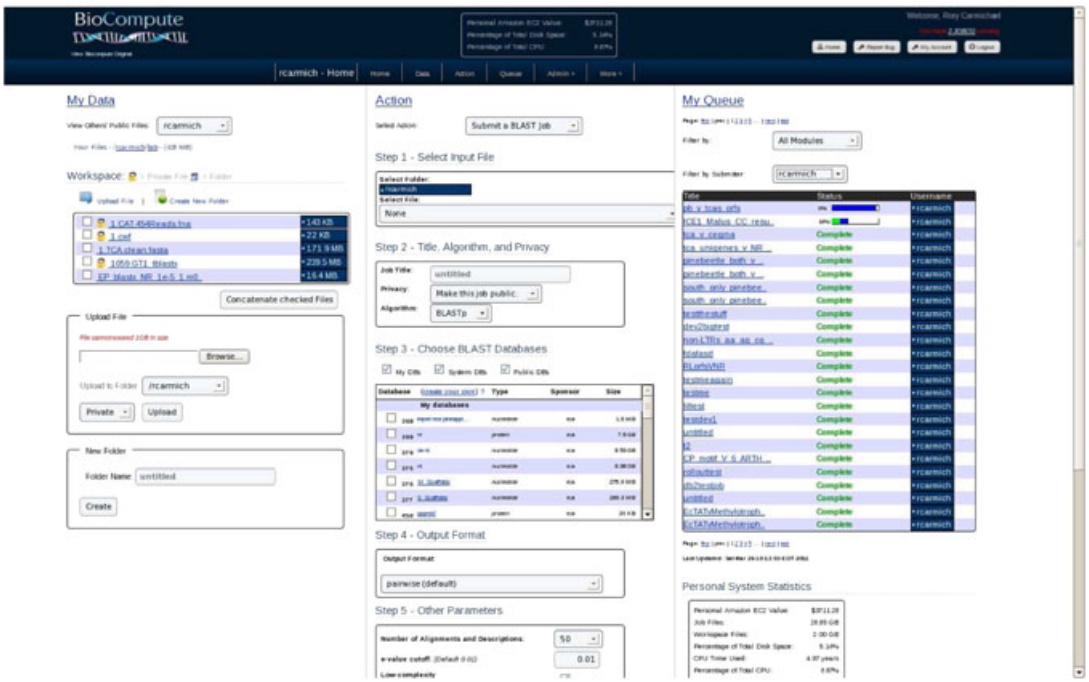

Figure 1. Biocompute interface.

DOI: 10.1002/cpe

4.1. Data

Users interface with their uploaded data much as they would in a regular filesystem; they can move files between folders, delete them, and perform other simple operations.

The same interface can be used to promote files to datasets (e.g., a BLAST reference database). To perform such a promotion, a user enters into a dialogue customizable by Biocompute tool developers. This screen permits users to set metadata as well as any required parameters. Once the selected file has been processed by the appropriate tool, the resulting data are distributed to a subset of the Biocompute Chirp [4] stores. The meta-information provided by the users is stored in a relational database, along with other system information such as known file locations. Our BLAST module uses this information to improve performance.

The data stored in Biocompute and the datasets generated from it are often elements of collaborative projects. User files and datasets may be marked public or private. A publically available dataset is no different from an administrator-created one, allowing users to respond to community needs directly. This primarily facilitates collaboration between biologists.

4.2. Actions: Biocompute modules

The initial version of Biocompute was a specialized tool for running parallel BLAST searches. To generalize it to multiple tools, we needed to provide application developers with simple hooks into the main site while providing enough flexibility for including diverse applications. Further, it was important that application developers be provided with a simple and flexible tool for describing and exploiting parallelism provided by the underlying distributed system.

From these requirements, modules emerged. Modules provide the core of biocompute's functionality. These tools cover a wide range in complexity, from SSAHA, a single executable, to SNPEXP, a complex set of Java programs. To support this variety, we developed a module structure that leverages encapsulation, interface consistency, record keeping, and data redundancy. By providing a simple mechanism for rapid development and deployment of parallel bioinformatics tools, we greatly facilitate collaborations between biologists and bioinformatics tool developers.

Conceptually, each module consists of an interface and an execution unit. The interface provides a set of PHP functions to display desired information to users via the web portal. The execution component is a template for the job directory used by Biocompute to manage batch jobs. So far, each Biocompute module utilizes a local script to generate and run a *Makeflow* [7] for execution on the distributed system. Most importantly, we have set up the system to allow developers to create modules without detailed knowledge of Biocompute. In fact, two of our available modules have been developed by computer science graduate students not otherwise involved with biocompute.

Namespacing provides much of the advantage of our modular structure. All of the elements required to run a module's job must be contained in a job template folder and follow a prescribed naming convention. In addition, each module must implement a common interface API in PHP. Essentially, modules inherit their characteristics from a generic model module (in fact we have implemented such a module to facilitate rapid development). A module is first instantiated by the biocompute architecture. This process is achieved by copying the template folder to a globally unique job number folder. This instance is initialized with the job's input and parameters. It is then executed through the standard wrapper executable provided in the job template, which in turn creates a makeflow specification of the work necessary and submits it to the appropriate batch system.

Makeflow [7] provides application developers needed simplicity and flexibility for developing distributed applications. A developer uses makeflow with make-like syntax to describe dependencies and execution requirements for their tool (see Figure 2 for an example makeflow). The fault-tolerant makeflow engine is capable of converting this workflow into the necessary submissions and management of distributed jobs on diverse batch systems. Whereas Biocompute's queue uses a relational database to store important job metadata—including job status, time and space utilization statistics, and input parameters—we are not satisfied with a single point of loss for this data. To manage the potential damage incurred by database loss or data corruption, we ensure that all job statistics can be

DOI: 10.1002/cpe

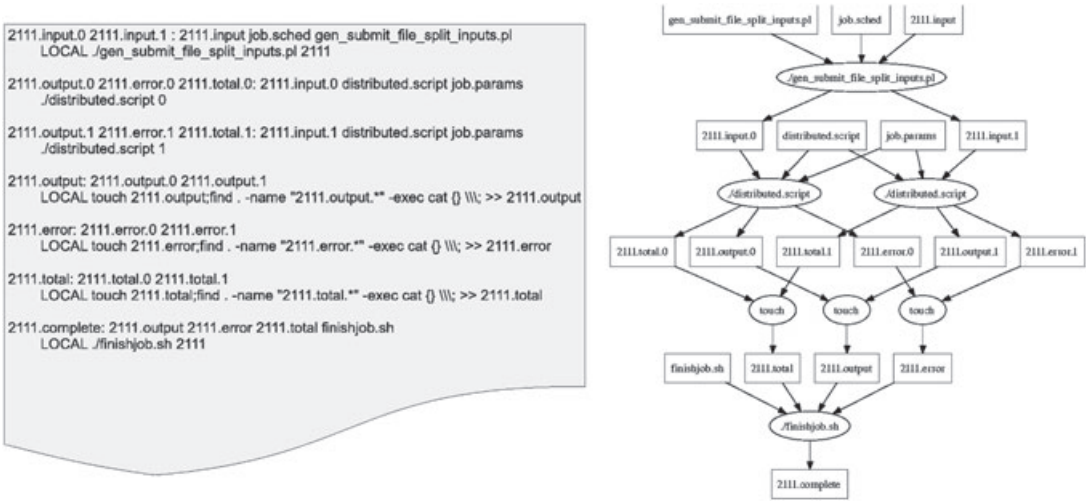

Figure 2. An example Makeflow file and corresponding graph of its execution. The distributed acyclic graph of this job is typical of Biocompute jobs.

recreated by inspecting the job directory. Each job records the execution plan specified by a makeflow file, a log of the timing and return values of the substeps in that execution, and detailed logs output by each individual executable and by the batch system selected for that job. It also freezes the input and output of the job so that future manipulations will not impact our ability to recover initial inputs and results.

4.3. Queue

Whereas the distributed system beneath Biocompute may not be of general interest to its users, the details of job submissions and progress are important. Biocompute provides methods for users to recall past jobs, track progress of ongoing jobs, and perform simple management such as pausing or deleting jobs. Users may view currently running jobs or look through their past jobs. Each job has drill down functionality, permitting a user to look through or download the input, output, and error of the job, and to see pertinent metadata such as job size, time to completion (or estimated time to completion if the job is still running), and parameters.

The queue facilitates collaboration in two primary ways. As with user data, queue jobs may be marked public or private. Making a job public exposes the source data, parameters, and results to inspection and replication by other users. Further, queue detail pages provide curious users with a way to evaluate the current resource contention in the system.

4.4. Data management for BLAST

BLAST [8], or Basic Local Alignment Search Tool, is a commonly used bioinformatics tool that implements a fast heuristic to align a set of one or more query sequences against a set of reference sequences. Because biologists often wish to compare hundreds of thousands of sequences against datasets containing gigabytes of sequence data, these jobs can take prohibitively long if executed sequentially. However, BLAST jobs are conveniently parallel in that the output of a single BLAST is identical to the concatenation of the output from BLASTs of all disjoint subsets of desired query sequences against the same reference dataset.

We found that using the dataset metadata stored in our relational database, we could rank machines by the number of bytes already pre-staged to that machine [1]. This rank function schedules jobs to the machine requiring the least dynamic data transfer. Whereas this approach naturally ties our BLAST module to Condor [5], similar concepts could be used to provide job placement preferences in other batch systems.

DOI: 10.1002/cpe

This technique requires that the list of dataset locations be correct. To maintain correct lists, we use the following method. Any successful job was run on a machine containing all required datasets. Jobs returning with the dataset not found error are known to lack one of the needed datasets. Therefore, we parse job results and update the relational database with data locations accordingly.

To balance load, we transfer databases from a randomly selected machine in the primary cluster. If the chosen machine lacks the appropriate database, the job fails and is rescheduled.

Even with random source selection load balancing, potential network traffic is significant. For example, the largest BLAST database hosted in Biocompute is over 8 GB. Further, many Biocompute jobs run in hundreds or even thousands of pieces, and therefore it is possible that a job could request 8-GB transfers to dozens or even hundreds of machines simultaneously. To mitigate this unreasonable demand, we limit the number of running jobs to 300. Additionally, we abort transfers that take longer than 10 min. This effectively prevents us from transferring files overly large for the available bandwidth.

4.5. Semantic comparison of manual and dynamic distribution models

The transition from manual to dynamic distribution required a shift in dataset storage semantics within Biocompute. This shift was brought about by the new authentication requirements introduced by node-to-node copying, and by the automatic updating characteristics provided by our logfile inspection technique. In Table I we document the characteristics of datasets before and after dynamic distribution. A replica is a copy of a Biocompute dataset. Primary machines are the core 32 machines described in the system description, over which we have direct administration privileges. Auditing scripts provide us with a means of manually determining the locations of dataset replicas in the cluster of primary machines. These were previously used to populate the database tracking those locations.

4.6. Performance of data management schemes

In this section, we will explore the performance characteristics of Biocompute, and evaluate the impact of data distribution model and data availability on these characteristics.

Table II illustrates the cost of the current timeout policy for the dynamic distribution model, as compared with a simple static distribution model. Figure 3 shows the runtime of a dynamic BLAST query executed against the NCBI non-redundant (NR) BLAST database. This dataset is 7.5GB - well outside the transferable size for our network. This effectively causes any job assigned to a machine without the database to immediately reschedule itself (in the static case) or to wait for some time and then reschedule (in the dynamic case). Although one might expect this behavior to significantly increase the overall runtime, we only observe an 18% increase in runtime for the dynamic distribution worst-case run. The characteristics of the rank function ensure that dynamic

Table I. Object characteristics before and after implementing dynamic distribution.

Object	Before dynamic		After dynamic	
	record	permissions	record	permissions
New primary machine replica	local fs	Biocompute, local	global	Biocompute, local, campus
Audited primary machine replica	global	Biocompute, local	global	Biocompute, local, campus
New dynamic replica	n/a	n/a	global	Biocompute, local

Table II. Worst case cost of dynamic distribution. Whereas the dynamic case shows a 167% increase in badput, it only suffers an 18% increase in runtime.

Distribution method	Execution time (h)	Badput (h)
Dynamic	17.09	517.3
Static	14.49	193.7

DOI: 10.1002/cpe

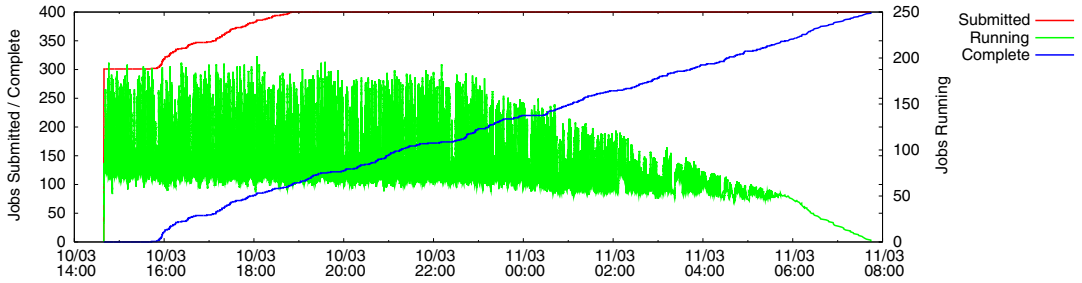

Figure 3. Worst case dynamic distribution run. The high variation in number of jobs running is a result of failed data transfer attempts.

jobs are only assigned to database-less machines when all of the usable machines are already occupied. In the static case, these jobs would simply wait in the queue until a usable machine became available. Essentially, the change to dynamic distribution forces a transition to a busy-wait. Whereas this increases badput—cycles wasted caused by job failures—and is suboptimal from the perspective of users competing with Biocompute for underlying distributed resources, it has a limited impact on Biocompute users.

Figure 4 shows the impact of varying timeout times on the runtime of a BLAST job. For this test, each database was initially distributed to the 32 core nodes and, depending on timeout, potentially distributed to any of the nodes open to Biocompute jobs. The lowest timeout time never transfers the target database, the middle one sometimes succeeds and sometimes fails, and the final one always succeeds. As expected, the increased parallelism generated by successful distribution reduces runtime. Long tail runtimes for some sub-jobs limited benefits from increased parallelism. A more sensitive input splitting scheme might mitigate this effect.

Our final timeout value was set high in order to maximize the transferrable database size, as the impact on performance was acceptable even for untransferable databases. Our final replication value was set to 32, the size of our dedicated cluster. Since the introduction of dynamic distribution, some datasets have been spread to up to 90 machines, tripling the parallelism available for those data. Many other datasets remain at or near their initial replication values, demonstrating that our distribution strategy has been sensitive to utilization levels of various datasets.

4.7. Social challenges of Biocompute

Up to this point, we have only addressed how Biocompute meets its technical challenges. However, as a shared resource, Biocompute requires several mechanisms to balance the needs of all of

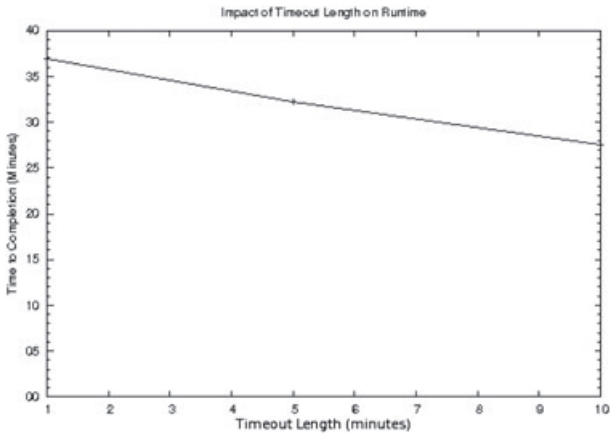

Figure 4. Runtime of BLAST versus a medium sized (491 MB) database for varying timeout lengths.

DOI: 10.1002/cpe

its stakeholders. We believe that this is best achieved through fair policies and transparency. In this section, we discuss the policies and tools we have used to facilitate the management of our two most limited resources: data and compute time.

The sheer size and number of available biological datasets require consideration of disk space costs of Biocompute. The two simplest ways of achieving this are motivating users to control their disk consumption, and convincing our users to provide or solicit funding to increase available space. In either case, it is necessary to provide users with both comparative and absolute measures of their disk consumption. To this end, we track disk space consumption and CPU utilization for all users, and publish a "scoreboard" within Biocompute. So far, Pie-charts have provided heavy users with sufficient motivation to delete the jobs and data they no longer need, and have given us insights into common use cases and overall system demand.

Resource contention also comes into play with regards to the available grid computing resources. Utilization of Biocompute's computational resources tends to be bursty, generally coinciding with common grant deadlines and with the beginnings and ends of breaks when users have extra free time. These characteristics have produced catastrophic competition for resources during peaks. Without a scheduling policy, job portions naturally interleaved, resulting in the wall clock time of each job to converge on the wall clock time necessary to complete all the jobs. To combat this problem, we implemented a first in first out policy for large jobs, and a fast jobs first policy for small (having at most five parts that can be executed in parallel) jobs. This has allowed quick searches to preempt long running ones, and prevents the activities of users from seriously impacting the completion timeline for previously started jobs.

4.8. Challenges revisited

Although our system mitigates issues of data management, display, and batch execution, challenges persist in the areas of permissions management and job state inspection.

The heterogenous environment upon which Biocompute is deployed complicates authentication and permissions. Biocompute users interact with multiple authentication systems such as website login, AFS (through Kerberos), Chirp, MYSQL, and Condor. Each system's permission model is optimized for its internal needs; these models often lack the flexibility needed by a meta-system incorporating diverse authentication and access control methods. The webhost submits and runs jobs on behalf of the user, which complicates bookkeeping.

To manage these challenges, we implemented metapermissions in the portal code. We track job and file ownership through the user's interactions with the website. Authentication with the rest of the systems is handled by machine-scoped permissions. However, this too complicates permissions management for remotely executing modules. Consider that permitting a module's sub-jobs the ability to intelligently update module specific tables at runtime, would require granting not only the module, but all distributed machines in the grid system, the ability to write to some elements of our system infrastructure. Whereas this would undeniably provide additional power to modules, it would also incur significant risk. A single sign-on solution might be appropriate if such a technology could be extended to our wide variety of tools.

It has proven surprisingly difficult to deduce the state of a batch job during its runtime. This is easy for the most common case (jobs are running successfully) because the logfile is regularly updated with increasing numbers of successful jobs. However, in our heterogenously distributed computing environment, we expect a moderate number of random failures. Jobs may fail because the remote machine may not possess the proper BLAST databases, because the owner of the machine evicts the job before it finishes, because of network failures or machine downtime, because of improper permissions (more common than one would hope for reasons discussed earlier), and many other reasons. It is important to distinguish between these failures, which impact only a particular subinstance of a job on a single machine, and systematic failures that guarantee that the job will not be completed. It seems likely that the trending of job progress, the evaluation of multiple jobs, and the real-time inspection of the detailed error information, which we collect might allow us to improve detection of true failures. Makeflow's provenance gathering functions would serve us particularly well in such an endeavor.

DOI: 10.1002/cpe

5. RELATED WORK

The provenance system at work in Biocompute bears similarity to the body of work generated by The First Provenance Challenge [9]. For biologists, the queue and detail views provide extremely high level provenance data. At the user interface level, we were most interested in giving the user an easily communicable summary of the process required to produce their results from their inputs. This model hides the complexity of the underlying execution, and gives our users a command that they can share with colleagues working on entirely different systems. Obviously, such information would be insufficient for debugging purposes, and Makeflow records a significantly more detailed record of the job execution, including the time, location and identity of any failed subjobs, and a detailed description of the entire plan of execution. Formal provenance systems are expected to have a mechanism by which queries can be readily answered [9]. We selected Makeflow as our workflow engine for its simplicity, and its ability to work with a wide variety of batch systems. A similarly broad workflow and provenance tool could replace it without modifying the architecture of Biocompute.

BLAST has had many parallel implementations [10, 11]. Whereas the BLAST module of Biocompute is essentially an implementation of parallel BLAST using makeflow to describe the parallelism, we do not mean to introduce Biocompute as a competitor in this space. Rather, we use BLAST to highlight common problems with the management of persistent datasets, and show a generic way to ease the problems generated without resorting to database segmentation or other complex and application-specific techniques. The BLAST module, like all of Biocompute's modules, uses an unmodified serial executable for each subtask.

Bioinformatics web portals such as Galaxy and Taverna provide biologists and bioinformaticians with tools to leverage existing bioinformatics web services in combination, and retain provenance data similar to that retained by Biocompute [12, 13]. However, these tools focus on leveraging existing web resources and reference datasets, and are limited by the portals to which they provide access. Biocompute focuses on providing users with the ability to powerfully query and share data unavailable in traditional portals.

6. CONCLUSIONS

We stated that our system should integrate user data, and make its management simple. Job parameters and meta-information should be kept and results easily shared and resubmitted. System resources should be scaleable and fairly, productively, and transparently shared.

Our system has, at least in pilot form, achieved these goals. However, our solutions are, for the most, part initial steps. By implementing the current version of Biocompute, we have exposed a need for a much more complex dataset management tool. With the addition of such a tool, Biocompute's ability to usefully classify user data and vary its storage and presentation policies based on these classifications could be expanded to any kind of data. At Notre Dame, in-house biological data generation is commonplace, and an effective system to pipeline this data directly into Biocompute would be very useful to our user community. Likewise, mechanisms to efficiently import data from online resources would be welcomed.

Biocompute addresses a broad spectrum of challenges in scientific computing web portal development. It illustrates some of the technical solutions to social challenges presented by collaborative and contentended resources, implements techniques for mitigating the performance impact of large semi-persistent datasets on distributed computations, and provides a useful framework for exploring and addressing open problems in cluster storage, computation, and socially sensitive resource management. Regular users range from faculty to support staff to students, and cover all areas of computational expertise from faculty to undergraduate biology majors; they span 10 research groups and three universities. In a year, we provided our users with more than 56 years of CPU time, and enabled them to perform research using custom datasets.

ACKNOWLEDGEMENTS

We would like to acknowledge the hard work of Joey Rich, Ryan Jansen, Thomas Potthaus, and Brian Kachmark who greatly assisted in the implementation of Biocompute. We would also like to thank Andrew

DOI: 10.1002/cpe

Thrasher, Irena Lanc, and Li Yu for their development of the SSAHA and SHRIMP modules. This work is supported by the University of Notre Dame's strategic investment in Global Health, Genomics and Bioinformatics and by NSF grant CNS0643229.

REFERENCES

1. Carmichael R, Braga-Henebry P, et al. Biocompute: towards a collaborative workspace for data-intensive bioscience. *WORKSHOP SESSION: Emerging Computational Methods for Life Sciences*, 2010.
2. Johnson M, Zaretskaya I, et al. NCBI BLAST: a better web interface. *Nucleic Acids Research* 2008; **36**:W5.
3. Lawson D, Arensburger P, et al. VectorBase: a home for invertebrate vectors of human pathogens. *Nucleic Acids Research* 2006; **35**:D503.
4. Thain D, Moretti C, et al. Chirp: a practical global filesystem for cluster and grid computing. *Journal of Grid Computing* 2009; **7**:51–72.
5. Litzkow M, Livny M, et al. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, 1998.
6. Raman R, Livny M, et al. Resource management through multilateral matchmaking. *International Symposium on High Performance and Distributed Computing*, 2000.
7. Yu L, Moretti C, et al. Harnessing Parallelism in Multicore Clusters with the All-Pairs, Wavefront, and Makeflow Abstractions. *Journal of Cluster Computing* 2010:1–14.
8. Altschul SF, Gish W, et al. Basic local alignment search tool. *Journal of Molecular Biology* 1990; **215**:402–410.
9. Moreau L, Ludascher B, et al. The first provenance challenge. *Concurrency and Computation: Practice and Experience* 2007; **20**:400–418.
10. Darling AE, Carey L, et al. The design, implementation, and evaluation of mpiBLAST. *Proceedings of ClusterWorld*, 2003.
11. Dumontier M, Hogue CWV. NBLAST: a cluster variant of BLAST for NxN comparisons. *BMC Bioinformatics* 2002; **3**:13.
12. Giardine B, Riemer C, et al. Galaxy: a platform for interactive large-scale genome analysis. *Genome Research* 3005; **15**:1451.
13. Oinn T, Addis M, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 2004.
14. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics* 2009; **25**:1754–60.
15. Ning Z, Cox AJ, et al. SSAHA: a fast search method for large DNA databases. *Genome Research* 2001; **11**:1725.
16. Rumble SM, Lacroute P, et al. SHRiMP: accurate mapping of short color-space reads. *PLoS Computational Biology* 2009; **5**:e1000386.

DOI: 10.1002/cpe