Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis

Shawn Hoon, Kiran Kumar Ratnapu, Jer-ming Chia, et al.

Genome Res. 2003 13: 1904-1915
Access the most recent version at doi:10.1101/gr.1363103

Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis

Shawn Hoon,[1] Kiran Kumar Ratnapu,[1] Jer-ming Chia,[2] Balamurugan Kumarasamy,[3] Xiao Juguang,[3] Michele Clamp,[4] Arne Stabenau,[5] Simon Potter,[4] Laura Clarke,[4] and Elia Stupka[3,6]

[1]Institute of Molecular and Cell Biology, National University of Singapore, Singapore 117609; [2]Genome Institute of Singapore, National University of Singapore, Singapore 117528; [3]Temasek Life Sciences Laboratory, National University of Singapore, Singapore 117604; [4]The Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SA, UK; [5]European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK

We identify several challenges facing bioinformatics analysis today. Firstly, to fulfill the promise of comparative studies, bioinformatics analysis will need to accommodate different sources of data residing in a federation of databases that, in turn, come in different formats and modes of accessibility. Secondly, the tsunami of data to be handled will require robust systems that enable bioinformatics analysis to be carried out in a parallel fashion. Thirdly, the ever-evolving state of bioinformatics presents new algorithms and paradigms in conducting analysis. This means that any bioinformatics framework must be flexible and generic enough to accommodate such changes. In addition, we identify the need for introducing an explicit protocol-based approach to bioinformatics analysis that will lend rigorousness to the analysis. This makes it easier for experimentation and replication of results by external parties. Biopipe is designed in an effort to meet these goals. It aims to allow researchers to focus on protocol design. At the same time, it is designed to work over a compute farm and thus provides high-throughput performance. A common exchange format that encapsulates the entire protocol in terms of the analysis modules, parameters, and data versions has been developed to provide a powerful way in which to distribute and reproduce results. This will enable researchers to discuss and interpret the data better as the once implicit assumptions are now explicitly defined within the Biopipe framework.

[Supplemental material is available online at www.genome.org. The Biopipe software operates under an open source license and is freely available at http://www.biopipe.org.]

The proliferation of bioinformatics journals and dedicated sections to bioinformatics in established journals are an indication that computational sequence analysis has become an integral part of biological studies (ISCB: http://www.iscb.org/journals.shtml). It is expected that the next generation of biologists will need to equip themselves with the necessary bioinformatics skills to complement their research (Zauhar 2001). A significant part of the programming work in bioinformatics involves writing "scripting glue," for purposes that include connecting different software applications together, file format conversions, and information extraction. It is for this reason that software toolkits like Bioperl (Stajich et al. 2002) have been developed to encourage software reuse by providing modular components to the programmer. Even with such toolkits, there is a fundamental deficiency in these analyses in providing the same level of rigorousness that is present in techniques in established fields like molecular biology. Protocol-driven lab techniques have made the ability to reproduce published results easily one of the hallmarks of scientific research. Protocols also make for easy benchmarks of different analysis techniques. The idea of applying protocols to bioinformatics analysis is not new. It has been more traditionally described as workflow management. Software projects like LabFlow (Goodman et al. 1998) and the proliferation of commercial products (http://www.limsource.com/products/vproduct.html) now on the market have sought to address the need for workflow management of large-scale biological research. They tend to focus more on laboratory information management systems (LIMS). These systems seek to manage experimental data and laboratory practices. Specifically, we seek to address protocols that are geared toward computational analysis. For example, one may define a protocol for gene building that is conservative with high confidence. However, if one is willing to have higher sensitivity at the expense of specificity, the protocol may be relaxed through the use of less stringent methods like building genes from ESTs (Hubbard et al. 2002). A protocol includes not only the definition of analysis and its parameters but also the data sets. It is the explicit definition of these protocols, their methodology and assumptions, that will not only allow researchers to interpret the data better but also allow these disparate tools to interoperate and be used synergistically to ask more complex questions.

Published bioinformatics data are easily available through Web site and ftp downloads. However, it is often more difficult when one is trying to replicate the methods of the analysis. Often, the reported methods may not fully capture the semantics and specific parameters that are used. Getting support from the author may be a tedious process as he or she may not package the software for easy installation or may already have moved on to different projects. In fact, it is often difficult even for the researcher when trying to replicate his own methodology on a new data set, for it involves deciphering code written earlier that may not be amenable to easy modifications. The problem stems from the nature of bioinformatics analysis, where the practice of writ-

[6]Corresponding author.
E-MAIL elia@tll.org.sg; FAX *65 6* 8727007.
Article and publication are at http://www.genome.org/cgi/doi/10.1101/gr.1363103. Article published online before print in July 2003.

13:1904–1915 ©2003 by Cold Spring Harbor Laboratory Press ISSN 1088-9051/03 $5.00; www.genome.org

ing "single use" scripts is common and the practice of writing clear, explicit documentation rare. This is because the methods are implicitly defined in the scripts and programs one writes. However, traversing through six months' worth of scripts that have minimal documentation and keeping track of program and data versions that were actually used in the analysis requires a Herculean effort. There is thus a clear need for a framework that will enable protocol-based bioinformatics analysis to be not only describable on paper but easily replicable and distributable as well. We argue that this creates greater transparency, encourages good scientific practices, and increases the rigorousness of bioinformatics analysis.

Data Fragmentation

The integrative nature of bioinformatics has made the ability to access different sources of data essential. The fragmented nature of bioinformatics databases has been metaphorically likened to the rival city-states of Pre-Renaissance Italy (Stein 2002). Without standardization of data access, it will remain vulnerable and risk descending into medieval chaos. Genome databases like Ensembl (http://www.ensembl.org), FlyBase (http://www.flybase.org), SGD (http://genome-ww.stanford.edu/Saccharomyces/), WormBase (http://www.wormbase.org), UCSC (http://genome.ucsc.edu), and NCBI (http://www.ncbi.nlm.nih.gov/) have been developed independently and are invaluable resources when accessed singularly. However, the different systems and formats on which they have been built pose a formidable challenge when one seeks to integrate data from these different sources for analysis. The inability to interoperate between various analysis tools compounds the problem through hindering the ability to ask more complex questions (Davidson et al. 1995). It is certain that integration of these disparate sources through formal Web service registries (Stein 2002) will be the future, but it will take considerable cooperation and time. Meanwhile, the symptomatic but very effective solution belongs to the list of open-source Bio* projects (http://www.open-bio.org), which provides various adaptor modules that enable the different data formats to be parsed and unified. We believe by harnessing these modules, one is able to develop an analysis system that is not tied to the data model on which it operates but instead affords data integration of different data sources and formats to carry out comparative analysis.

Bioinformatics Pipelines

It is quite certain that with the increasing number of genomes that are being sequenced over the next few years, computational biologists will have an embarrassment of choices for the amount of data available for analysis. A classification of tasks in bioinformatics emphasizes that most bioinformatics requirements may be described in terms of filters, transformers, transformer-filters, forks, and collections of data (Stevens et al. 2001). Two themes are consistent in these requirements: (1) the need for running analysis in a serial rule-dependent fashion (workflow) and (2) the ability to run these tasks in parallel where possible (high-throughput).

The most immediate evidence of this was the development of sequence assembly tools and genome annotation systems to deal with the tsunami of data (Roos 2001) pouring from the sequencing factories. These forms of analysis often operate under a compute farm model, that is, an environment in which multiple CPUs, memory, and storage facilities are linked via special software to provide massive parallel computational performance with great cost effectiveness. The costs of hardware for such systems are rapidly decreasing, and it is expected that most small labs will have some sort of compute farm available to them, which may be as simple as a series of unused desktops strung together.

Despite the short history of bioinformatics, several software efforts have evolved into a pipeline model. The Labflow system (Goodman et al. 1998) is a Perl-based object-oriented framework aimed at providing LIMS capabilities and includes an engine for executing workflows. It is built on top of a commercial relational database management system that stores information necessary for workflow tracking. The limited adoption of this system is a result of the immaturity of bioinformatics software at the time. As a result, it was not able to provide richness in its functionality and thus utility as well. Pise (Letondal 2002) is a Web interface generator for molecular biology programs aimed at providing an easy interface for usage while allowing programs to be combined in a pipelined fashion. This system is easy to use and highly popular but is not meant for automated large-scale analysis. In principle, these pipeline systems channel data through a series of analyses. A notable example that does this while working in a parallel fashion through execution of jobs over a compute farm is the Ensembl Annotation Pipeline (Hubbard et al. 2002). It has been used successfully to annotate the *Danio rerio*, *Drosophila melanogaster*, *Takifugu rubripes*, *Homo sapiens*, *Anopheles gambiae*, and *Mus musculus* genomes (http://www.ensembl.org; http://www.fugubase.org). A similar pipeline was used in the Gadfly Annotation project (Mungall et al. 2002). Such automated or semiautomated pipelines have proven to be robust systems for managing bioinformatics data at the application layer and are a model that we propose will be important for future bioinformatics analysis.

Yet Another Pipeline?

It is important to justify why we see the need for another system that follows the pipeline model in view of many previous efforts that have not garnered widespread adoption. It is not because of design or implementation issues. Indeed, most of these projects were well thought out, and Biopipe has adopted many of the same concepts. Instead, we feel that the level of functionalities in the software components available in the bioinformatics community at that time had not reached a critical mass for which an overarching framework may be applied. Over the last couple of years, there have been several initiatives in the Open Source community to apply rigorous software engineering practices to bioinformatics software in an effort toward interoperability among the many projects (Chicurel 2002). We strongly feel that it bodes well now to develop a system that is able to harness and anticipate the functionalities that these initiatives will bring about. To this end, we have identified the need for a robust system that will allow the conduct of large-scale computational analysis in a reproducible, rigorous, and flexible manner. The Biopipe system that we have developed is an effort working toward such a goal. In collaborative work done with the Ensembl Team, we have worked to redesign the Ensembl pipeline into a more generic system for bioinformatics analysis. It is a framework developed in the Perl language and is aimed to be flexible in allowing researchers to develop pipelines through reusable modules. It is flexible in the handling of data sources in anticipation of the ever-evolving state of data models and formats, a situation innate to bioinformatics. In addition, by inheriting the Ensembl Pipeline's model for working in a compute farm environment, Biopipe's job management system is able to plug in different load-sharing management systems based on the user's specific hardware environment. Through this framework, each pipeline is easily configurable by way of analysis parameters and is easily reproducible by others through a common exchange format. We propose that a pipeline

and its configurations may be thought of as a protocol, a well-defined way of conducting a specific analysis.

RESULTS

The Biopipe project is hosted by the Open Bioinformatics Foundation (OBF; http://www.open-bio.org). See Methods for a list of system requirements necessary for running Biopipe. For convenience, a tarball of all packages required to run Biopipe as well as the example pipelines working with BioSQL and Ensembl are available at http://www.biopipe.org/bioperl-pipeline-download.html and will be updated regularly. Moreover, the live CVS tree of the source code may be accessed at http://cvs.bioperl.org/cgi-bin/viewcvs/viewcvs.cgi/bioperl-pipeline/?cvsroot=bioperl.

Taking the perspective of the end-user, we first describe how the Biopipe framework can be used to structure the design process of a bioinformatics protocol. We then describe in more detail the fundamental design concepts behind the Biopipe framework.

Designing Bioinformatics Protocols

The Biopipe framework aims to allow researchers to focus on their specific biological analysis and avoid having to deal with issues like data access and parsing and job management. These problems have already been solved through various software modules, and what is clearly needed now is the framework in which to plug them together. A series of steps toward designing a pipeline in Biopipe may be described as follows: (1) Design the analysis protocol. (2) Incorporate implementation design factors. (3) Choose appropriate analysis modules and configure parameters. (4) Write XML describing the protocol, sources, and modules.

We use a phylogenetic pipeline as an example to take through this process. To determine the relationship between classes of sequences, phylogenetic analysis is often essential. Our particular problem involves the analysis of distantly related protein sequences from the Fugu fish, human, and fly. These sequences are assumed clustered into protein families. A semantic description is as follows:

Given sets of related protein sequences, determine their phylogenetic relationships.

Step 1. Design the Analysis Protocol

This step deals with the design concept of the experiment to be carried out. It involves framing the questions that one wants to answer through the protocol and deciding on the type of data needed to answer the questions. It also involves thinking about the quality of data that is available and the data sanitization process that may need to be carried out before the analysis. This is also a good place to state the assumptions and possible drawbacks with this particular protocol. In other words, we seek to address the questions this protocol can answer as well as the ones that it cannot. The result of this is a well-defined qualitative description of the overall pipeline with an explicit statement of goals and assumptions.

For our use case, a simple example of such a description may be as follows:

Goal

Given sets of related protein sequences grouped together in families, carry out a series of analyses (Fig. 1A) that result in phylogenetic trees. This allows one to infer paralogous and orthologous relationships. Fly proteins are assumed to be sufficiently distant to function as outgroups. We want to be able to use different programs and parameters to see the variability of the trees generated.

Data Sets

- Human protein sequences are taken from the Ensembl annotation database (http://www.ensembl.org). We screen for proteins from known genes only.
- Fugu protein sequences are taken from an in-house-curated set of Fugu genes.
- Fly protein sequences are taken from the Berkeley *Drosophila* Genome Project (http://www.bdgp.org; Adams et al. 2000.

Assumptions

- We assume a constant rate of evolution in our set of protein sequences.
- Bootstrapping replicates of multiple alignments will provide sufficient statistical significance of the phylogenetic trees generated.

Step 2. Incorporate Implementation Design Factors

We have defined the "scientific" aspect of the protocol. Now we need to design the pipeline implementing that protocol. The main consideration at this point is to define the data unit on which the job (single unit of analysis) will operate. This is important for it addresses how we will break up our analysis into separate chunks so that we can execute them in a parallel fashion. At the same time, we define the format and location of our input and output sources.

For our use case, this translates to:

Data Unit: A family of protein sequences.

Input: There are three different input sources: (1) Human protein sequences reside in the Ensembl database and may be accessed through the Ensembl API. (2) Fugu protein sequences are stored in a BioSQL database and may be accessed via the bioperl-db API. (3) Fly protein sequences are stored in an indexed Fasta formatted file and may be accessed through the bioperl package.

Output: The final results are PostScript files containing the phylogenetic trees for visualization. In addition, intermediate results are stored as files for future reference (Fig. 1B).

Step 3. Choose Appropriate Analysis Modules and Configure Parameters

We now select the specific analysis modules that we want to use. For each pipeline stage defined in step 1, we may have more than one program to choose from. This choice is made depending on the nature of the questions to be addressed and the data sets on which the analysis operates. For example, when considering the program to be used for multiple protein alignment, it has been shown that for more distantly related protein sequences, the T-Coffee (Notredame et al. 2000) alignment program is more sensitive than CLUSTALW (Thompson et al. 1994). Alternatively, one may select a different PAM substitution matrix instead. This is a decision to be made based on one's specific biological question. It is up to the pipeline designer to select the modules and parameters that best suit the particular protocol. A designer may also choose to experiment with different programs by benchmarking against well-verified data sets. The analysis parameters are also configured at this point, and this may also be subject to experimentation (Fig. 1C).

Step 4. Write XML Describing the Protocol, Sources, and Modules

After the preceding steps, the pipeline design process is completed. Next, the pipeline is implemented through writing the XML file. This XML file would serve as the complete description of the protocol of the data experiment that is being performed. Subsequent changes to the protocol need only be reflected in the XML file.

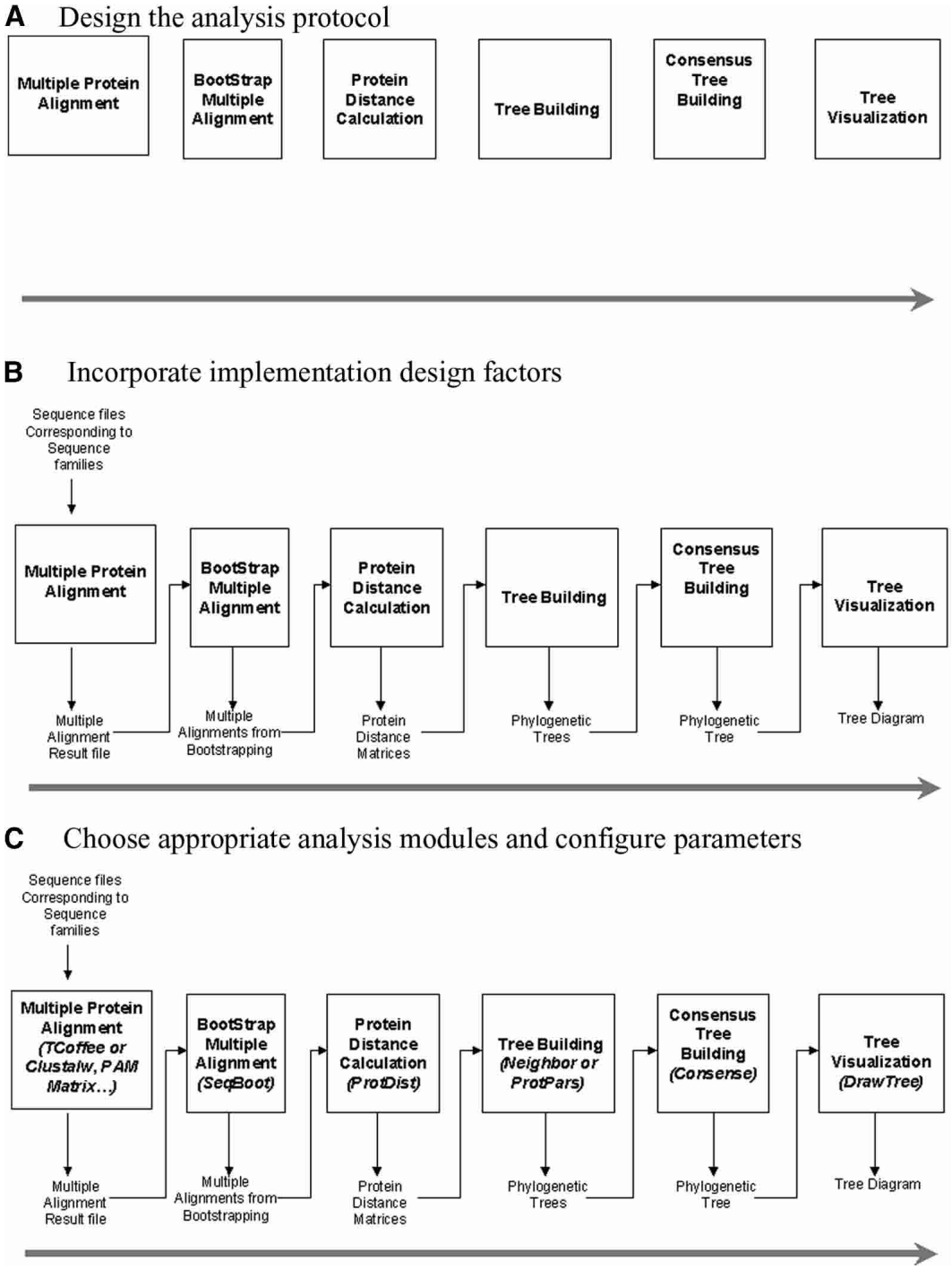

Figure 1 (Continued on next page)

An excerpt of the phylogenetic pipeline XML is shown in Figure 1D.

The Biopipe Design

The focus of Biopipe's design is principled on two considerations: (1) robustness and flexibility and (2) reproducibility.

Robustness and Flexibility Through Database Persistency

Biopipe strives to manage parallel jobs over a cluster of computational nodes. It achieves this through a database centric design. The state and specification of the pipeline is stored in the Biopipe database at any given time. The Biopipe database works off an mySQL (http://www.mysql.com) Relation Database Management System (RDMS). A control script called PipelineManager queries this database via various Biopipe modules to access the information necessary to run the pipeline. The type of information accessed by the control script may be grouped under two areas, pipeline specification and pipeline state.

During the course of running a large number of jobs, system failure, hardware or software, may necessitate stopping the pipe-

D Write XML describing the protocol, sources and modules

```
<pipeline_setup>
    <pipeline_flow_setup>
          <!-- This specifies the analysis and rules of the pipeline. Analysis  refer to the runnables tha
         will be used in this pipeline while the rules specify the order in which these analysis are
         to be run, including any specific pre-processing actions that are to be carried out. -- >

       <analysis id="2">
           <logic_name>TCoffee</logic_name>
           <runnable>Bio::Pipeline::Runnable::MSA</runnable>
           <program>align</program>
           <program_version> 1.37</program_version>
           <analysis_parameters>-ktuple 2 -matrix PAM 250</analysis_parameters>
           <runnable_parameters>-program TCoffee -infile_dir t/data/phylip_dir </runnable_parameters>
           <output_iohandler id="1"/>
       </analysis>
                                .
                                .

       <rule>
          <current_analysis_id>2</current_analysis_id>
          <next_analysis_id>3</next_analysis_id>
          <action>COPY_ID_FILE</action>
       </rule>

    </pipeline_flow_setup>
                                .
                                .
    <data_source_setup>
        <!-- This specifies the datasources that the pipeline will need to run
        on and the modules needed to read/write data from these data sources. -- >
         <streamadaptor id="1">
             <module>Bio::Pipeline::Dumper</module>
         </streamadaptor>
                                .
                                .
    </data_source_setup>

    <iohandler_setup>
        <!- This specifies the series of method calls on a data_source module to read/write the data. -- >
        <iohandler id="1">
            <adaptor_id>1</adaptor_id>
            <adaptor_type>STREAM</adaptor_type>
            <iohandler_type>OUTPUT</iohandler_type>
            <method>
            <name>new</name>
            <argument>
                <tag>-dir</tag>
                <value>t/data/phylip_result</value>
            </argument>
            <argument>
                <tag>-file_suffix</tag>
                <value>cls</value>
            </argument>
            <argument>
                <tag>-prefix</tag>
                <value>INPUT</value>
                <type>SCALAR</type>
            </argument>
            <argument>
                <tag>-module</tag>
                <value>generic</value>
            </argument>
            <argument>
                <tag>-format</tag>
                <value>phylip</value>
            </argument>
        </method>
        <method>
          <name>dump</name>
          <argument>
              <value>OUTPUT</value>
              </argument>
        </method>
     </iohandler>
                                .
                                .
    </iohandler_setup>

</pipeline_setup>
```

Figure 1 (*A–C*) An example of the steps involved in the design of a bioinformatics protocol for phylogenetic analysis. (*D*) Excerpt of the phylogenetic tree-building pipeline XML.

line. Diagnostic information may be accessed through the Biopipe database. Once the fault is rectified, the pipeline can continue from where it left off by rerunning the control script because the pipeline state is stored in the database.

In addition, having the entire specification of the pipeline in the database provides great flexibility in allowing different input and output sources to be "programmed" at the database level rather than the API level. This is a feature that makes it easy to add or remove input sources on existing pipelines with minimal code modifications.

Pipeline Specification

The pipeline specification refers to the pipeline configuration that describes the particular analysis protocol. This is made up of the following properties: (1) the list of analysis programs involved and the order in which the analyses are executed; and (2) a description of where input data resides, how to retrieve it, and where to output the results of the analysis. The analysis component and the I/O component encapsulate these two properties, respectively.

Analysis Component

The analysis component is designed for modularity in enabling different analysis programs to be run in a workflow setting. Information exchange is achieved through the use of various parsing modules found in Bioperl that enable data to be represented commonly as objects. The source of data and their formats are abstracted from the pipeline via these objects. As long as the analysis layers recognize these objects, they can be run on the pipeline. Bioperl already has modules for parsing a wide range of data formats, modules for accessing remote databases like GenBank and EMBL, and parsers for many program outputs that all return standard objects on which Biopipe may operate.

This is a result of the Biopipe software layer that wraps the actual analysis program and defines an interface that enables it to be plugged into the pipeline (Fig. 2). This layer consists of three components, parsers, wrappers, and runnables.

Parsers: Different analysis programs provide different output formats to represent the same biological entity. For example, many sequence analysis programs return sequence features, which are landmarks of interest found on a piece of DNA sequence that have a start, end, and strand. However, formats chosen to represent them in program outputs may be ad hoc and often depend on the author's sense of aesthetic appeal. To become independent of these formats, we use Bioperl objects to represent them in software. For example, output from various gene prediction programs are represented as Bio::SeqFeature::Gene::GeneStructure objects. With this common representation, these genes can then be manipulated in the same manner. However, before we can represent them as objects, we need parsers that interpret each specific format to extract the pertinent information. A host of parsers for various bioinformatics programs have been written under the Bio::Tools::* name space as part of the core Bioperl package bioperl-live. These parsers are written as stand-alone modules that may be used by scripts or as in our case the wrappers described below.

Wrappers: Wrappers are software modules that provide an interface to executable programs. It is a common software engineering practice that allows for information hiding by providing a common interface through which to access the underlying program. These wrappers also allow for different input types like objects, files, or file handles and allows for greater flexibility in usage. In Bioperl, many wrappers have been written as part of the bioperl-run package. Biopipe makes use of these wrappers to encapsulate the instantiation and execution of programs like BLAST (Altschul et al. 1990; W. Gish, http://blast.wustl.edu), CLUSTALW (Thompson et al. 1994), and Genscan (Burge and Karlin 1997). We have separated the design of the wrappers from Biopipe itself such that these wrappers are in effect stand-alone modules that may be used by scripts. These wrappers can be found in the bioperl-run distribution at the Bioperl Web site (http://www.bioperl.org). An important design decision that we made was not to restrict the way people write their wrappers, and we want to be able to plug in wrappers external to Bioperl as well. The way we do this is through the Biopipe runnables described in the next point.

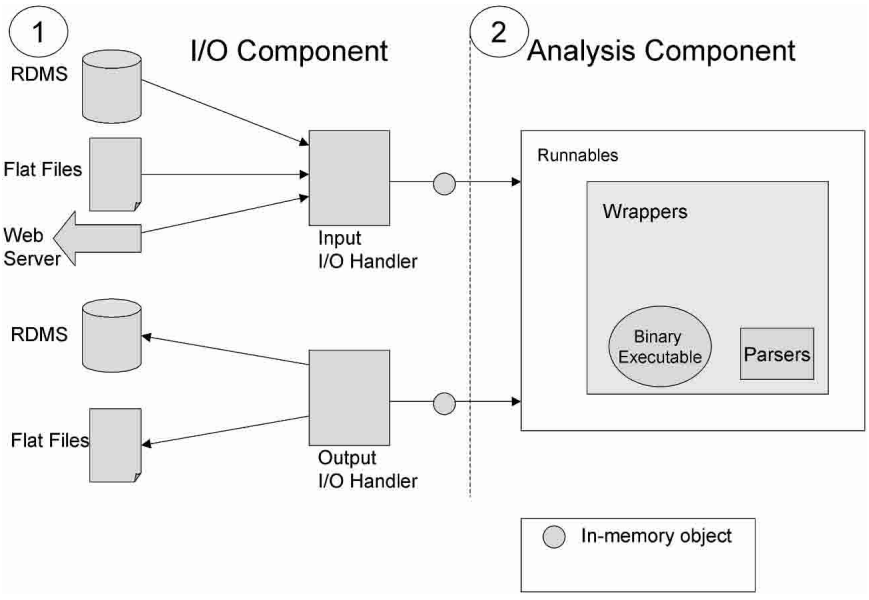

Figure 2 (1) The I/O component of Biopipe. The design of I/OHandlers allows different input and output sources to be plugged in for analysis. (2) The modular breakdown of the Analysis component. Inputs fetched via the I/OHandlers are passed to the Analysis component as in-memory objects. Wrapper and parser modules reside outside of Biopipe, and the modular design allows for different wrappers and parsers to be swapped into the runnable, which acts as an interface to the Biopipe system. The modular design allows for swapping of different wrappers and parsers into the runnable.

Runnables: To allow different programs to be plugged into the pipeline, we allow a further layer of abstraction that accommodates different wrappers with different interfaces. We do this with a set of pipeline modules called runnables, listed in Table 1, that encapsulate different wrappers depending on how wrappers are written. This gives a layer of abstraction between the pipeline and the analysis layer while providing flexibility as to how wrappers may be written. For example, we have a PHYLIP (Felsenstein 1983) runnable that plugs a set of PHYLIP wrappers into Biopipe. This runnable encompasses programs like consense, drawgram, drawtree, neighbor, protdist, protpars, and seqboot.

I/O Component

The I/O Component is designed in anticipation of the diverse nature of bioinformatics data. One of the main features of Biopipe

Table 1. Biopipe Runnables

Modules (Bio::Pipeline::Runnable::*)	Class
BLAST	General alignment
Blat	Fast sequence alignment
DNA Block Aligner	Sensitive DNA alignment
Eponine	Transcription start site finder
Est2Genome	EST to genomic sequence alignment
Fasta	Fasta suite of alignment programs that include fasta, tfastx-fasty
Genewise	Homology-based gene prediction
Genscan	Ab initio gene prediction
Molphy	Molecular phylogenetics package
MSA	Multiple sequence alignment programs like CLUSTALW and Tcoffee
PAML	Phylogenetic analysis by maximum likelihood package
PHYLIP	PHYLIP package programs that include consense, drawgram, drawtree, neighbor, protdist, protpars, and seqboot
Primate	Short oligo search
Promoterwise	Sensitive alignment for promoter sequences
ProteinAnnotation	Suite of protein annotation programs that includes Seg, Coils, Hmmpfam, FingerPrintScan, PFScan, Signalp, Tmhmm
Pseudowise	Pseudo gene predictor
RepeatMasker	Interspersed repeat finder
Sim4	CDNA to genomic sequence alignment
TribeMCL	Protein family clustering

A snapshot of the list of runnables available in Biopipe. These runnables serve as an interface between the wrappers and the pipeline system.

is to allow analysis to run on disparate input sources. This includes more complex representations like genes, phylogenetic trees, or protein family clusters as well as more traditional inputs like DNA and protein sequences. We foresee that as bioinformatics analysis evolves, we will need to deal with more complex biological representations like expression data, regulatory elements, and so on. We do so through creating a Biopipe object called I/OHandler that encapsulates the entire logic necessary to fetch or store data. An I/OHandler object is made up of two objects, Adaptors and DataHandlers.

Adaptors: Adaptors are interfaces between the database and its underlying schema and the application layer. We have two types in Biopipe: database and stream adaptors. We refer to database adaptors as modules that provide a connection to a Relational Database Management System (RDBMS). We refer to stream adaptors as modules that allow data to be fetched from any source other than a RDBMS. This includes file-based and remote Web servers. All interactions with the data source are made through these adaptors without any knowledge of the specifics in their underlying design. Once the data are fetched and represented in memory as objects, these objects may be manipulated in the same way. In this way, Biopipe allows different input sources to be used for the same analysis.

DataHandlers: DataHandlers are a Biopipe representation of the method calls that are made to the adaptors. DataHandlers may also have arguments.

Example: For example, programmatically in the Ensembl API, a Bio::EnsEMBL::Gene object is fetched via the following snippet of code:

```
#this line creates a dbadaptor object which provides a handler to the
#database called 'homo_sapiens' using the 'mysql' driver on a host server 'pulse' and using user id 'ensembl'
my $db = Bio::EnsEMBL::DBSQL::DBAdaptor->new(-dbname=>"homo_sapiens",
  -user   =>"ensembl",
  -driver =>"mysql"
  -host   =>"pulse");
#this line gets the gene adaptor that fetches a Bio::EnsEMBL::Gene

#object using a stable id
my $gene = $db->get_GeneAdaptor->fetch_by_stable_id('ENSG000000001');
```

In Biopipe terminology, this translates to:

Adaptor: **B**io::EnsEMBL::DBSQL::DBAdaptor
Adaptor parameters: -dbname=>"homo_sapiens"

–user =>"ensembl",

–host =>"pulse"
DataHandler 1: get_GeneAdaptor
DataHandler 2: fetch_by_stable_id
DataHandler 2 Argument: 'ENSG000000001'

These two DataHandlers are thought of as a cascade of methods with get_GeneAdaptor having precedence over fetch_by_stable_id and are executed in that order. The DataHandler argument is associated with the second DataHandler in this example. The Adaptors and DataHandlers for output handling are similar.

Through this design, we are able to represent in Biopipe almost any Perl-object-oriented logic for fetching and storing data. In this way, the IOHandler is able to work with the main sources of input and output that we expect to be prevalent in bioinformatics: relational databases (input and output); flat files (input and output); remote Web servers (only input); and OBDA.

Relational Databases: We anticipate most bioinformatics data to be stored in varied relational database schemas. Two good examples of such systems are the BioSQL (http://obda.open-bio.org/) and Ensembl databases. The former is a sequence database that started as an open-source alternative to the Sequence Retrieval System (http://srs.ebi.ac.uk/) and is rapidly evolving into a robust system. The latter is a genome annotation storage facility with well-developed APIs for data manipulation and has an excellent Web component for display. They are implemented with adaptors that provide a consistent method in retrieval and storage of data through the passing of objects. Any database may be connected to the Biopipe framework as long as

appropriate database adaptors exist. In this way, we can allow different objects to interact from different database sources. For example, one can read in contig sequences from Ensembl and align them against various sequences from BioSQL or GenBank formatted flat files.

Flat Files: To read data from files (for example Fasta, GenBank, and EMBL formatted sequence files), we use stream adaptors that harness the suite of file-indexing modules already written in Bioperl, such as the Bio::Index and the Bio::DB::Flat systems. These modules build indices on files in various formats for quick random access, allowing one to treat a flat file as a database. Programmatically, data from files are fetched in a similar fashion to data from relational databases. For example, to read files from a Fasta formatted file indexed using Bio::Index::Fasta, one would use the following code:

```
#create an index object from the fasta.index file
my $inx = Bio::Index::Fasta->new('-filename'= "fasta.index");
#fetch the sequence using id "104K_THEPA"
my $seq = $inx->fetch("104K_THEPA");
```

In Biopipe terminology, this translates to

Adaptor: **B**io::Index::Fasta
Adaptor parameters: -filename=>"fasta.index"
DataHandler 1: new
DataHandler 2: fetch
DataHandler 2 Argument: '104_THEPA'

Remote Web Servers: Stream adaptors allow data to be read from remote sources using modules already developed within the Bioperl project. This is often not advisable when building a large bioinformatics workflow because of the unreliability of remote protocols. However, it has been included to allow users who do not have large local storage resources to run small pipelines against public databases. This could be useful, for example, if researchers want a simple analysis repeated regularly on their sequence of interest. In that case, it would be also important to monitor version numbers on the sequences retrieved to validate results properly. For example, the module Bio::DB::GenBank fetches sequences remotely from NCBI's GenBank database and may be plugged into the Biopipe as a data source:

```
my $db = Bio::DB::GenBank->new( );
my $seq = $db->get_Seq_by_id('AE003439');
```

Adaptor: **B**io::DB::GenBank
DataHandler 1: new
DataHandler 2: get_Seq_by_id
DataHandler 2 Argument: 'AE003439'

Other sources include

- Bio::DB::EMBL
- Bio::DB::Fasta
- Bio::DB::GenPept
- Bio::DB::RefSeq
- Bio::DB::SwissProt
- Bio::DB::XEMBL.

Because of performance issues involved in transferring from the Internet, the Web-based database adaptors work best with smaller data sets.

OBDA: The Open-Bio OBDA project has already endeavored to mask some of the complexity in retrieving sequences from all of the above systems. This is achieved by implementing a Registry system that allows seamless retrieval of sequences by their accession numbers regardless of the underlying data source. In Bioperl the registry is used via the Bio::DB::Registry module, which may be plugged in similarly to the examples above as an adaptor in conjunction with *new* and *get_database* DataHandlers.

Transformers: Transformers are Biopipe's representation of modules that may be attached to IOHandlers for preprocessing of data before they are passed to the analysis component and postprocessing of data before results are stored in the output database. Two main classes of data transformers in Biopipe are Filters and Converters. A significant amount of logic in analyzing data involves filtering. The filtering criteria logic may be based on something as simple as a BLAST score or involve complex heuristics specific to a particular analysis. Present filters include filtering of sequence features based on score, length, and a variety of other criteria. We expect a wide variety of criteria to be in place. It is for this reason that we provide a flexible scheme in which to plug in different filtering logic that people can write by adhering to a simple interface. At the same time, objects fetched through a particular API may not be compatible to a Bioperl runnable. It is for this reason that we have Converters modules, which extract information from a particular object model and map it to another. The modular nature of these transformer modules enables them to be chained as demonstrated in Figure 3.

Pipeline State

The pipeline state is maintained via the Job Management System. The Biopipe Job Management System is derived from the Ensembl Annotation Pipeline (Hubbard et al. 2002), a system that has proven robust in a high-throughput environment.

The Job Unit: The Biopipe Job Management System revolves around the job unit (Fig. 4A). This unit may be thought as a self-contained unit of analysis that is able to fetch its own inputs, run the analysis on its inputs, and store the results. Each job is associated with a single analysis and has the following diagnostic properties: status, stage, standard out and error file, and retry count.

Status: During the course of its execution, the job maintains a certain state. The states presently available include *NEW*, *SUBMITTED*, *FAILED*, and *COMPLETED*. A *NEW* job is one that has been newly created. A *SUBMITTED* job is one that has been batched to the compute farm. A *FAILED* job is one that has failed during the course of execution. A *COMPLETED* job is one that has executed successfully. Each job is responsible for updating its status to the central Biopipe Database. It is through this central registry that the control script PipelineManager makes decisions on submitting jobs, rerunning failed jobs, and suspending jobs that have failed repeatedly.

Stage: When a job is in *SUBMITTED* or *FAILED* status, it may be in one of three

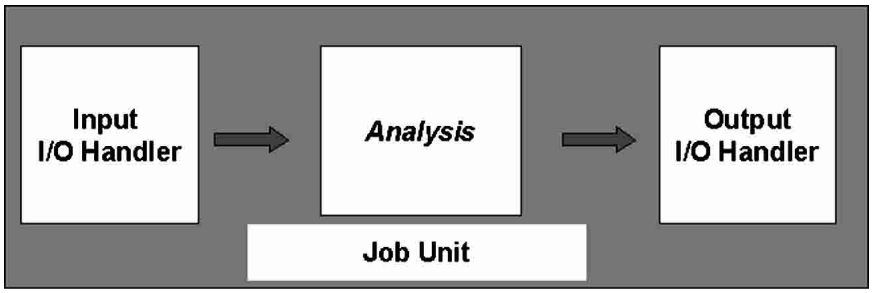

Figure 3 Transformers are applied to inputs after they are fetched from input I/OHandlers. They are also applied before being passed to the output I/OHandler. Transformers are modular in nature and may be chained.

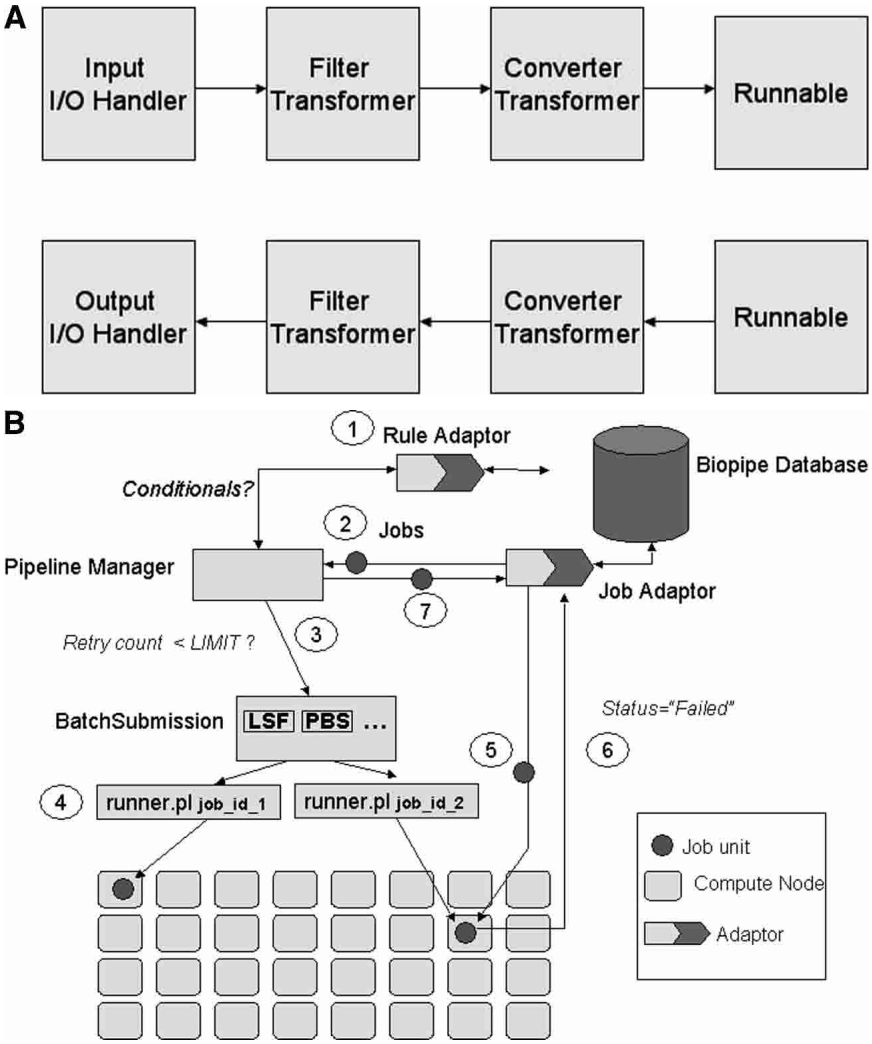

Figure 4 (*A*) A single job unit, made up of three components. The input I/Ohandler tells the job how and where to fetch the input for the analysis. The analysis component tells the job what to run on the input. The output I/Ohandler tells the job how and where to store the results of the analysis. (*B*) (1) Rules are fetched via the Rule Adaptor when PipelineManager starts up. (2) A Job Unit/object is fetched via the JobAdaptor. Only Jobs with status *NEW* or *FAILED* are considered for job submission. (3) Jobs are submitted via the Batch Submission object. *FAILED* jobs are checked against a retry limit before being submitted. (4) The mechanism for job submission: A runner.pl script is passed along with the job ID to the underlying load-sharing software that dispatches the execution of the script to one of the compute nodes. (5) At the node, the job is recreated by fetching the job unit via the JobAdaptor using the job ID. (6) The job updates its running status during execution. (7) The PipelineManager creates new jobs for subsequent analysis according to rules and conditionals.

possible stages: *READING*, *RUNNING*, or *WRITING*. This corresponds to the three components of the job unit: Input IOHandler, Analysis, and Output IOHandler, respectively. This provides an added level of information as to what part of the job is running during its execution.

Standard Out and Error File: Each job has standard out and error files that capture the output of the program execution. This contains the most descriptive diagnostic information pertaining to any execution errors that might arise.

Retry Count: Each job that fails will be retried. The retry count keeps track of the number of times it has been rerun. A retry count limit may be set on the number of times to rerun a fail job.

Job Management: The batching of jobs to a farm of computational nodes provides parallel performance but yields complexity in keeping tracking of what jobs are running, failed, and what to run next in the pipeline. Thus the Job Management component (Fig. 4B) of Biopipe is critical. This logic is implemented in the PipelineManager control script, and the important components in this design are (1) rules definition and (2) the job fetching and submission mechanism.

Rules Definition: Jobs are batched according to a workflow that we define as rules in Biopipe. These rules define the order in which jobs are to be created. When a job is completed, the PipelineManager control script will decide, based on the rule definition, the analysis to run next. If there is a following analysis, it will create the next job according to the certain rule conditionals. These conditionals include *WAITFORALL*, *COPY_INPUT*, and *NOTHING*. They tell the system what to do with a job once it is completed. A *WAITFORALL* conditional defines that the job has to wait for all jobs of the same analysis to finish before proceeding to the next analysis. A *COPY_INPUT* conditional specifies that the next job to be created for the next analysis will reuse the same input object. A *NOTHING* conditional tells the system not to create a new job.

Job Fetching and Submission: Jobs with status *FAILED* or *NEW* are fetched as Biopipe Job objects from the Biopipe Database via the JobAdaptor. *FAILED* jobs are checked against a retry limit before being submitted. Submission is achieved by the PipelineManager through the BatchSubmission module. This module is a wrapper around the underlying load-sharing software that manages the jobs over a cluster. The softwares presently supported by Biopipe are LSF (http://www.platform.com) and PBS (http://www.openpbs.org/). They have both proven to be robust and optimized for resource allocation in a distributed environment. The underlying mechanism by which jobs are submitted in Biopipe is simple. A script called runner.pl along with the Biopipe job IDs are submitted to the load-sharing software. In turn, the execution of the script is then dispatched to one of the compute nodes available. The script, when executed on the remote node, recreates the job in memory by querying the central Biopipe Database. It is able to do so because the pipeline specification is persistent in the database. Once submitted, the job will update its own status in the pipeline as *SUBMITTED*.

Once a job is completed or has failed, it will also update its status appropriately. Based on this information, the PipelineManager script will make decisions, with reference to the rules, on submitting new jobs, rerunning failed jobs, or suspending jobs that have failed repeatedly.

Reproducibility Through a Common Exchange Format

The second consideration in Biopipe's design, which is key to its adoption, is the degree of ease with which to reproduce the protocols. It became clear that the degree of flexibility we incorpo-

A Loading the Pipeline using XML

```
Terminal — tcsh — tcsh (ttyp1)
[Shawn-Hoons-Computer:~/cvs_src/bioperl-pipeline] shawn% perl scripts/PipelineManager -dbname phylo_
pipe -l -norun -xml xml/templates/phylip_tree_pipeline.xml -schema sql/schema.sql
Creating phylo_pipe
   Loading Schema...
Reading Data_setup xml   : xml/templates/phylip_tree_pipeline.xml
Doing DBAdaptor and IOHandler setup
Doing Transformers..
Doing Pipeline Flow Setup
Doing Analysis..
Doing Rules
Doing Job Setup...
Loading of pipeline phylo_pipe completed
PipelineManager exited
[Shawn-Hoons-Computer:~/cvs_src/bioperl-pipeline] shawn%
```

B Running the Pipeline

```
Terminal — tcsh — tcsh (ttyp1)
[Shawn-Hoons-Computer:~/cvs_src/bioperl-pipeline] shawn% perl scripts/PipelineManager -dbname phylo_
pipe -l -f
Removing Lock File...
Done
////////////////////////////////////////////////////////////
7 analysis found.
Running test and setup..

////////////// Analysis Test //////////////
Checking Analysis 1 DataMonger
-------------------- WARNING ----------------------
MSG: Skipping test for DataMonger
---------------------------------------------------
 ok
Checking Analysis 2 Clustalw ok
Checking Analysis 3 Phylip ok
Checking Analysis 4 Phylip ok
Checking Analysis 5 Phylip ok
Checking Analysis 6 Phylip ok
Checking Analysis 7 Phylip ok
Fetching Jobs...
Fetched 1 incomplete jobs
Running job /tmp/phylo_pipe_DataMonger.1047226269.54.out /tmp/phylo_pipe_DataMonger.1047226269.54.er
r
Fetched 1 completed jobs
Going to snooze for 3 seconds...
Waking up and run again!
Fetching Jobs...
Fetched 1 incomplete jobs
Running job /tmp/phylo_pipe_Clustalw.1047226281.42.out /tmp/phylo_pipe_Clustalw.1047226281.42.err
```

...truncated for space...

```
Writing plot file ...
Finished.
End of run.
Going to snooze for 3 seconds...
Waking up and run again!
Fetching Jobs...
Fetched 0 incomplete jobs
Fetched 1 completed jobs
Going to snooze for 3 seconds...
Waking up and run again!
Nothing left to run.

///////////////Shutting Down Pipeline/////////////////////
Removing Lock File...
Done
////////////////////////////////////////////////////////////
[Shawn-Hoons-Computer:~/cvs_src/bioperl-pipeline] shawn%
```

Figure 5 (Continued on next page)

rated into the Biopipe's database persistent design would result in a great number of configuration settings. This would make it complicated, even for developers, to configure a pipeline via SQL. More importantly, we wanted a way in which end-users would be able to exchange pipeline configurations and install the pipeline easily. We also wanted a structured way in which the entire protocol of the analysis could be defined. This would also allow users to further experiment with the pipeline by modifying parameters or swapping different analysis modules. It was for these reasons that we adopted XML as an exchange format. XML is becoming the de facto standard for structuring documents with a high degree of flexibility and ease of usage. It is also used in an increasing number of bioinformatics projects (http://www.xml.com/pub/rg/Bioinformatics).

We have designed an XML DTD that defines all the necessary components for running a pipeline. Each pipeline will result in an XML document. This pipeline is loaded into the Biopipe Database via a simple script. A more detailed description of the XML DTD may be found in the Biopipe documentation available at the Biopipe Web site (http://www.biopipe.org). See also Supplemental Material (available online at www.genome.org) for more information about the XML DTD.

DISCUSSION

The motivation behind the Biopipe project was identified during the authors' experience in trying to carry out bioinformatics analysis that involved multispecies analysis in various collaborative projects with other groups. We found ourselves wanting to mix different input objects like protein sequences with DNA sequences, protein families with phylogenetic trees, and so on. We found that the usage of scripts to glue different software components together did not scale well in terms of protocol design or high-throughput analysis. Instead, it would often require rewriting a significant amount of software for similar analysis. This, in turn, made it extremely difficult to define the exact methodology of our methods and to streamline processes in an analysis environment not amenable to such large-scale cross-species comparisons. As a result, ad hoc software becomes more the norm than the exception. Such methods are vulnerable to data format changes and program interface variations. These negative aspects, in turn, hamper productive work in a collaborative environment. Unless methodologies are recorded and easy to reproduce (i.e., exchangeable in a defined protocol exchange format), collaborators are not able to replicate them because certain details of parameters and data sources may have been omitted or changed over time. This then often leads to confusion as to why a collaborator has not achieved the same results, and as to whether it is because of the methodology or the specific operating environment. Most importantly, in this tedious process, the specifics of the biological motivations behind the analysis are often lost in the details of the implementation, and bioinformaticians often spend more time solving engineering problems than looking at the data and the results.

Our formulation of Biopipe's design was a result of the melding of ideas from two open-source bioinformatics software projects: Bioperl and Ensembl. Our experience with the Ensembl Pipeline Annotation System was important in providing us the model in which to formulate Biopipe's database persistent design. Ensembl's open-source philosophy allowed us to incorporate a significant amount of design features into the Job Management System of Biopipe. The database persistent design is robust and well tested in a production environment, having been used to annotate a number of genomes where jobs that number in the hundreds of thousands at any one time are handled easily.

At the same time, we found that although the pipeline worked well when it dealt solely with the Ensembl Database, it was very difficult when we wanted to run it on different data sources and protocols other than genome annotation. It was not easy to plug in different analysis modules for bioinformatics experimentation. Bioperl's rich set of modules inspired us to design a system that worked in a similar fashion to Ensembl but extended to allow disparate data sources and modules to be inte-

C Checking the Job status

```
Terminal — ssh — tcsh (ttyp2)
shawnh@pulse1 ~/src/bioperl-pipeline/scripts> perl job_viewer  -dbname zfish_blast_pipe

BIOPIPE JOBVIEWER 0.1

|Analysis Available from zfish_blast_pipe|
 +----+-------------+
 | id |  Analysis   |
 +----+-------------+
 | 1  | DataMonger  |
 | 2  | Blast1      |
 | 3  | Blast2      |
 +----+-------------+

Select an analysis to get view job status [1-3] [1] 2

|Analysis Blast1|
 +-----------+---------+---------+---------+---------+-------+
 |  Status   | READING | WRITING | BATCHED | RUNNING | TOTAL |
 +-----------+---------+---------+---------+---------+-------+
 | NEW       | 0       | 1       | 676     | 0       | 677   |
 | SUBMITTED | 0       | 0       | 148     | 0       | 148   |
 | FAILED    | 26      | 0       | 0       | 0       | 26    |
 | COMPLETED | 0       | 0       | 0       | 0       | 26045 |
 +-----------+---------+---------+---------+---------+-------+

Jobs that failed belong to the following stages:
 +----+----------+
 | id |  STAGE   |
 +----+----------+
 | 1  | READING  |
 +----+----------+

Select a stage to view a sample err file for failed jobs of analysis Blast1[1-4] [1]
```

Figure 5 A simple example of a Biopipe session. (*A*) Loading the pipeline using the XML template. (*B*) Running the pipeline using PipelineManager. (*C*) Checking the Job Status via the Job Viewer script.

grated for different analysis protocols. The three main features of Bioperl that we used are

- Biological object representations (sequences, genes, families, alignments, etc.)
- Analysis program parsers (BLAST, Genewise, etc.)
- Application wrappers (BLAST, Genscan, PHYLIP, etc).

The object-oriented nature of Bioperl made it easy to exchange information between analysis modules using objects as a broker. This is made possible by the numerous parsing modules that enable files to be converted into other formats. The application wrappers provide a level of abstraction by providing an interface between the executable programs and Biopipe runnables. Biopipe runnables provide the final abstraction between wrappers and Biopipe. With this abstraction, it is now possible to swap different analysis modules in Biopipe as long as the inputs to these analysis modules are available.

Biopipe is designed to be generic where possible in anticipation of the evolving nature of bioinformatics analysis. The abstraction of input and output handling together with the modular approach to analysis components should accommodate a significant number of analysis protocols. The result of such flexibility is the number of options and parameters available. Such complexity makes setting up and designing pipelines a daunting task. We have designed an XML DTD aimed to provide a standard format in which to exchange pipeline protocols. The result of this XML design is twofold: (1) provide an easy way in which to exchange and replicate protocols; (2) document thought processes by forcing researchers to define explicitly every aspect of their protocol into a single document.

One of the present limitations of Biopipe is the diversity of analysis modules that it supports. Naturally, the types of analysis that are presently available are mainly implemented in relation to our research interests. However, we expect this to change as we expect that the open-source nature of Biopipe and its affiliation with the other OpenBio projects will mean that researchers will contribute their XML and software to this project. Moreover, Biopipe was initiated primarily as a software engineering project, and as such the initial concerns were to achieve a good architecture and design. Thus, it is presently targeted to users with experience in PERL and Bioperl. It would be very interesting now that the core design has been accomplished to develop an interface that is more user-friendly. This would allow end-users to run complex pipelines seamlessly. One could imagine, for example, allowing the users to upload data files and give an intuitive graphical interface for designing pipelines and monitoring their status. We have explored the idea of a stand-alone package to design pipelines and find that it provides an overly complex interface for the end-user. We are now exploring the idea of a Web interface that would allow simple modifications to pre-established protocols.

In the future, we seek to develop Biopipe in the following ways:

- Develop more analysis modules and XML-defined protocols that include ortholog detection, noncoding sequence comparisons, pseudogene prediction, and motif searching.
- Develop better rule handling for pipeline workflow to allow explicit user intervention and more sophisticated conditionals.
- Refine XML design to allow seamless concatenation of pipelines.
- Create more user-friendly interfaces for pipeline design.
- Create better interfaces for Job Tracking and Management.

In summary, Biopipe is a flexible framework that aims to allow researchers to focus on protocol design. The ability to plug in different modules and tune parameters makes for easy bioinformatics experimentation. A common exchange format that encapsulates the entire protocol in terms of the analysis modules, parameters, and data versions is a powerful way in which to distribute and reproduce results. We argue that this lends rigorousness to bioinformatics analysis and will enable researchers to discuss and interpret the data better once the implicit assumptions are explicitly defined.

METHODS

System Requirements

Biopipe requires the following software components:

1. The Perl interpreter, version 5.6.0 or higher (http://www.perl.org).
2. The MySQL database management system, version 3.23.43 or higher (http://www.mysql.org).
3. A load-sharing software, either LSF (http://www.platform.com) or PBS (http://www.openpbs.org/).
4. The following packages, all of which are hosted at the Bioperl Web site (http://www.bioperl.org):
   - bioperl-pipeline—the Biopipe software modules
   - bioperl-run—Perl wrappers for various binary executables
   - bioperl-live—the core Bioperl package
5. Binary executables. This is dependent on the type of analysis modules one is using for their pipelines. This information is stated explicitly in each pipeline XML document, and more information is available at the Biopipe Web site (http://www.

biopipe.org). For our phylogenetic example, the list of binaries required is:

- CLUSTALW (ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalX/)
- TCoffee (http://igs-server.cnrs-mrs.fr/~cnotred/Projects_home_page/t_coffee_home_page.html)
- PHYLIP Package (http://evolution.genetics.washington.edu/phylip.html)

6. Installation details are available at the Biopipe Web site (http://www.biopipe.org).

Running Biopipe

Running complex bioinformatics analysis in a high-throughput manner does not come without engineering challenges. Our goal is to ameliorate some of these difficulties with Biopipe so that the user can concentrate on the scientific questions that he or she wants to answer and not be bogged down by technical issues. However, the design of a compute cluster presents major challenges that are beyond the scope of this document. It involves considerations that are specific to one's hardware architecture. As such, issues such as location of databases, file systems, and other optimization tweaks may vary, and the goal of Biopipe is to accommodate these different configurations as much as possible. Hence, we do expect some level of technical knowledge from our end-users. We envision that the average Biopipe user should be fairly conversant with bioinformatics analysis and have basic programming and database management skills. Users should also work closely with the compute farm administrator so as to configure Biopipe to suit their specific environment.

To get Biopipe running, the user or system administrator must first ensure that the various packages and executable programs are installed properly. Also, the various databases and file systems must be configured properly for access. The second step is to create a Biopipe database using the schema provided in the bioperl-pipeline package. Next the user will configure the pipeline XML file to define his or her input and output data sources, analysis parameters, and so on. Finally, he or she can run the pipeline using the PipelineManager control script. During the course of execution, the user may query the status of his or her jobs via the Job Viewer script. A sample session of running the phylogenetic pipeline is shown in Figure 5. A list of the pipeline templates available at present may be found at the Biopipe Web site (http://www.biopipe.org).

ACKNOWLEDGMENTS

The Biopipe Core presently comprises, in alphabetical order, Jerming Chia, Shawn Hoon, Kiran Kumar Ratnapu, Balamurugan Kumarasamy, Elia Stupka, and Xiao Juguang. The authors acknowledge contributions from the following people, in alphabetical order: Chen Peng, Aaron Chuah, Low Yi Jin, Tania Oh, Andy Nunberg, Jason Stajich, Lenny Teytelman, and Frans Verhoef. The authors are also especially grateful to Lincoln Stein for his incisive comments during the preparation of this manuscript. We thank Brian Osborne for help in providing documentation for the project. We also acknowledge the work done by the Ensembl and Bioperl community on which this project is largely based. The Biopipe project is supported under the umbrella of the Open Bioinformatics Foundation (http://www.open-bio.org).

The publication costs of this article were defrayed in part by payment of page charges. This article must therefore be hereby marked "advertisement" in accordance with 18 USC section 1734 solely to indicate this fact.

REFERENCES

Adams, M.D., Celniker, S.E., Holt, R.A., Evans, C.A., Gocayne, J.D., Amanatides, P.G., Scherer, S.E., Li, P.W., Hoskins, R.A., Galle, R.F., et al. 2000. The genome sequence of *Drosophila melanogaster*. *Science* **287:** 2185–2195.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. Basic local alignment search tool. *J. Mol. Biol.* **215:** 403–410.

Burge, C. and Karlin, S. 1997. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.* **268:** 78–94.

Chicurel, M. 2002. Bioinformatics: Bringing it all together. *Nature* **419:** 751–755.

Davidson, S.B., Overton, C., and Buneman, P. 1995. Challenges in integrating biological data sources. *J. Comput. Biol.* **2:** 557–572.

Felsenstein, J. 1983. PHYLIP (Phylogeny Inference Package) version 3.5c. Distributed by the author. Department of Genetics, University of Washington, Seattle, WA.

Goodman, N., Rozen, S., and Stein, L.D. 1998. The LabFlow system for workflow management in large scale biology research laboratories. *Proc. Int. Conf. Intell. Syst. Mol. Biol.* **6:** 69–77.

Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T., et al. 2002. The Ensembl genome database project. *Nucleic Acids Res.* **30:** 38–41.

Letondal, C. 2002. A Web interface generator for molecular biology programs in Unix. *Bioinformatics* **17:** 73–82.

Mungall, C.J., Misra, S., Berman, B.P., Carlson, J., Frise, E., Harris, N., Marshall, B., Shu, S., Kaminker, J.S., Prochnik, S.E., et al. 2002. An integrated computational pipeline and database to support whole-genome sequence annotation. *Genome Biol.* **3:** research0081.1–0081.1.

Notredame, C., Higgins, D.G., and Heringa, J. 2000. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* **302:** 205–217.

Roos, D.S. 2001. Computational biology. Bioinformatics—Trying to swim in a sea of data. *Science* **291:** 1260–1261.

Stajich, J.E., Block, D., Boulez, K., Brenner, S.E., Chervitz, S.A., Dagdigian, C., Fuellen, G., Gilbert, J.G., Korf, I., Lapp, H., et al. 2002. The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.* **12:** 1611–1618.

Stein, L. 2002. Creating a bioinformatics nation. *Nature* **417:** 119–120.

Stevens, R., Goble, C., Baker, P., and Brass, A. 2001. A classification of tasks in bioinformatics. *Bioinformatics* **17:** 180–188.

Thompson, J.D., Higgins, D.G., and Gibson, T.J. 1994. CLUSTALW: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* **22:** 4673–4680.

Zauhar, R.J. 2001. University bioinformatics programs on the rise. *Nat. Biotechnol.* **19:** 285–286.

WEB SITE REFERENCES

ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalX/; CLUSTALW.
http://blast.wustl.edu; BLAST.
http://cvs.bioperl.org/cgi-bin/viewcvs/viewcvs.cgi/bioperl-pipeline/?cvsroot=bioperl; live CVS of the Biopipe source code.
http://evolution.genetics.washington.edu/phylip.html; PHYLIP Package.
http://genome.ucsc.edu; genome database.
http://genome-ww.stanford.edu/Saccharomyces/; *Saccharomyces cerevisiae* genome database.
http://igs-server.cnrs-mrs.fr/~cnotred/Projects_home_page/t_coffee_home_page.html; TCoffee.
http://obda.open-bio.org/; sequence database generated as open source alternative to the Sequence Retrieval System.
http://srs.ebi.ac.uk; Sequence Retrieval System.
http://www.bdgp.org; Berkeley *Drosophila* Genome Project database.
http://www.bioperl.org; Bioperl Web site, Biopipe wrappers.
http://www.biopipe.org; Biopipe.
http://www.biopipe.org/bioperl-pipeline-download.html; a tarball of all packages required to run Biopipe.
http://www.ensembl.org; Ensembl Annotation database for human protein sequences.
http://www.flybase.org; *Drosophila* genome database.
http://www.fugubase.org; Fugu genome database.
http://www.iscb.org/journals.shtml; journals dedicated to bioinformatics.
http://www.limsource.com/products/vproduct.html; on proliferation of commercial products for workflow management.
http://www.mysql.com; MySQL Database Management System.
http://www.ncbi.nlm.nih.gov/; NCBI genome database.
http://www.open-bio.org; Open Bioinformatics Foundation.
http://www.openpbs.org/; PBS software.
http://www.platform.com; LSF software.
http://www.wormbase.org; *Caenorhabditis elegans* genome database.
http://www.xml.com/pub/rg/Bioinformatics; XML.
http://www.perl.org; Perl software.

Received March 27, 2003; accepted in revised form May 21, 2003.