BioFilter: An Architecture for Parallel Deployment and Dynamic Chaining of Standalone Bioinformatics Tools.

Thomas Brettin and Avinash Kewalramani
Genomic Sequencing and Computational Biology Group (B-5), Bioscience Division
Los Alamos National Laboratory. Bikini Atoll Road, SM 30,Los Alamos, NM 87545
brettin@lanl.gov, avinash@lanl.gov

Abstract

A large number of bioinformatics analysis tools available today are processor intensive. Keeping in mind that the amount of biological data to be analyzed is growing steadily, and these tools are not easily deployable on Beowulf clusters, we propose BioFilter. BioFilter is a software implementation of an architecture that provides a framework for the deployment of these tools on a cluster thereby providing large performance increases. Another facet of bioinformatics analysis is that the tools, in general, are not used in isolation, but are used collaboratively during an analysis. Our architecture provides a way to create pipelines of different analysis tools chained together in a cluster environment, passing the output from one analysis tool as the input to another analysis tool.

1. Introduction

Informatics based biological research is dependent on many computational software tools that differ depending on the research areas, i.e. genome sequencing, genome annotation, structure prediction, etc. However, the basic idea is that each software tool transforms data from one form to another. That data is then analyzed for vital information or passed along as input to the next computational tool. Consider a genome analysis team that sets up analysis pipelines to annotate a genome. A typical analysis pipeline might include a gene prediction tool like Glimmer [1,2] to predict the genes in the genome DNA sequence and a tRNA prediction tool like tRNAscan [3] to predict tRNA genes. Similarity searches on these genes could involve BLAST [4] and the output used as is or passed to other parsing tools for taxonomic analysis. The gene sequences from the gene prediction tool could serve as input to secondary structure prediction tools like Psort [5], and Coils [6], be analyzed for complexity using Seg [7,8], or used with protein domain search tools like Prodom [9], Blocks [10,11] and Prosite [12]. The underlying point is that genome analysis involves a lot of data streaming between tools which if done manually would be a waste of time for the annotator and would waste disk space when storing intermediate data. Finally, in a research environment, the structure of the analysis pipeline is not fixed but is dynamic due to the very nature of scientific inquiry.

The architecture we describe has two advantages: i) the pipeline allows one to automate and reconfigure workflows easily; ii) the parallel aspect provides for performance acceleration. Performance acceleration might be improved more than what we have achieved when methodology like MPI is used. However, such methodologies do introduce unnecessary complexity. Tradeoffs are required to efficiently address complexity and performance. Furthermore, the ease with which a new tool can be introduced into the architecture played an important role in our decisions.

The BioFilter architecture addresses the aspect of dynamic analysis pipelines and automatic data streaming by using the Pipes and Filters architecture pattern [13, 14]. The basic idea of the Pipes and Filters pattern is *"Objects that have compatible interfaces but perform different transformations and computations on data streams can be dynamically connected to perform arbitrary operations"* [13]. The tools in the analysis pipeline are modeled as objects in the Pipes and Filters pattern. This enables the user to create dynamic pipelines with automated data streaming.

Parallelization of CPU intensive bioinformatics tools can be achieved by splitting the input data set and running these tools as multiple services on the Beowulf cluster. BioFilter achieves parallelization by using the Broker architectural pattern [14] to structure and coordinate services. To implement this architecture we use the Client-Dispatcher-Server [14] pattern during the design stage. The server runs the computational tool and since there are multiple servers running on different nodes, each computational tool is duplicated. The client manages splitting of the input data set and plays the role of an object with a compatible interface in the Pipes and Filters pattern. The dispatcher acts as an intermediate layer between the Client and the Server providing location transparency by means of a name service, and hides the details of the connection between clients and servers.

Thus the BioFilter architecture turns the cluster into a data flow computer, based on the concept of data-driven computation as defined in [18]. Each pipeline is analogous to a data flow graph, with each cluster processor corresponding to a node in this graph. Every node carries out some operation based on availability of data. The specific operation carried out by each node and the routing of data between nodes is all part of the software implementation of the architecture.

responsibilities and include rules and guidelines for organizing the relationships between them [14]. The dominant architectural patterns for BioFilter are the Pipes and Filters pattern and the Broker pattern. Pipes and Filter pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component with data being passed through pipes between adjacent filters. Recombination of these filters facilitates building families of related systems. Because

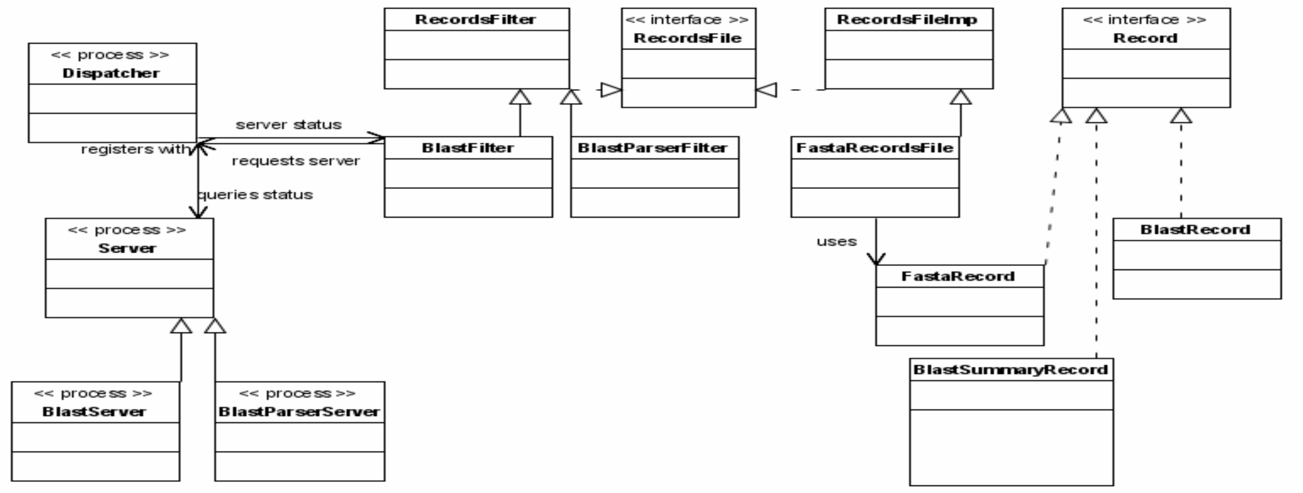


Fig 1. Class diagram of the BioFilter architecture

The BioFilter architecture is unique in that it does not attempt to take the existing tools and alter their algorithms for parallel execution. This type of approach is tool-specific and can be slow in terms of development time. Therefore, attempting to modify the tools is not feasible as the number of tools to parallelize increases. Parallelization in BioFilter is achieved by duplication of computational tools allowing easy and seamless integration of many bioinformatics tools into a cluster environment. Since the original computational tools remain unaltered after being plugged into the BioFilter architecture, the output of the tool is the same as if it were run on a workstation in its original form. The BioFilter architecture is in no way constrained by the cluster hardware configuration, the kind of operating system or the queuing system running on the cluster. Our implementation utilizes the forking capability of UNIX system, the PERL programming language, and a Beowulf cluster with shared disk resources.

2. Architectural and Design patterns

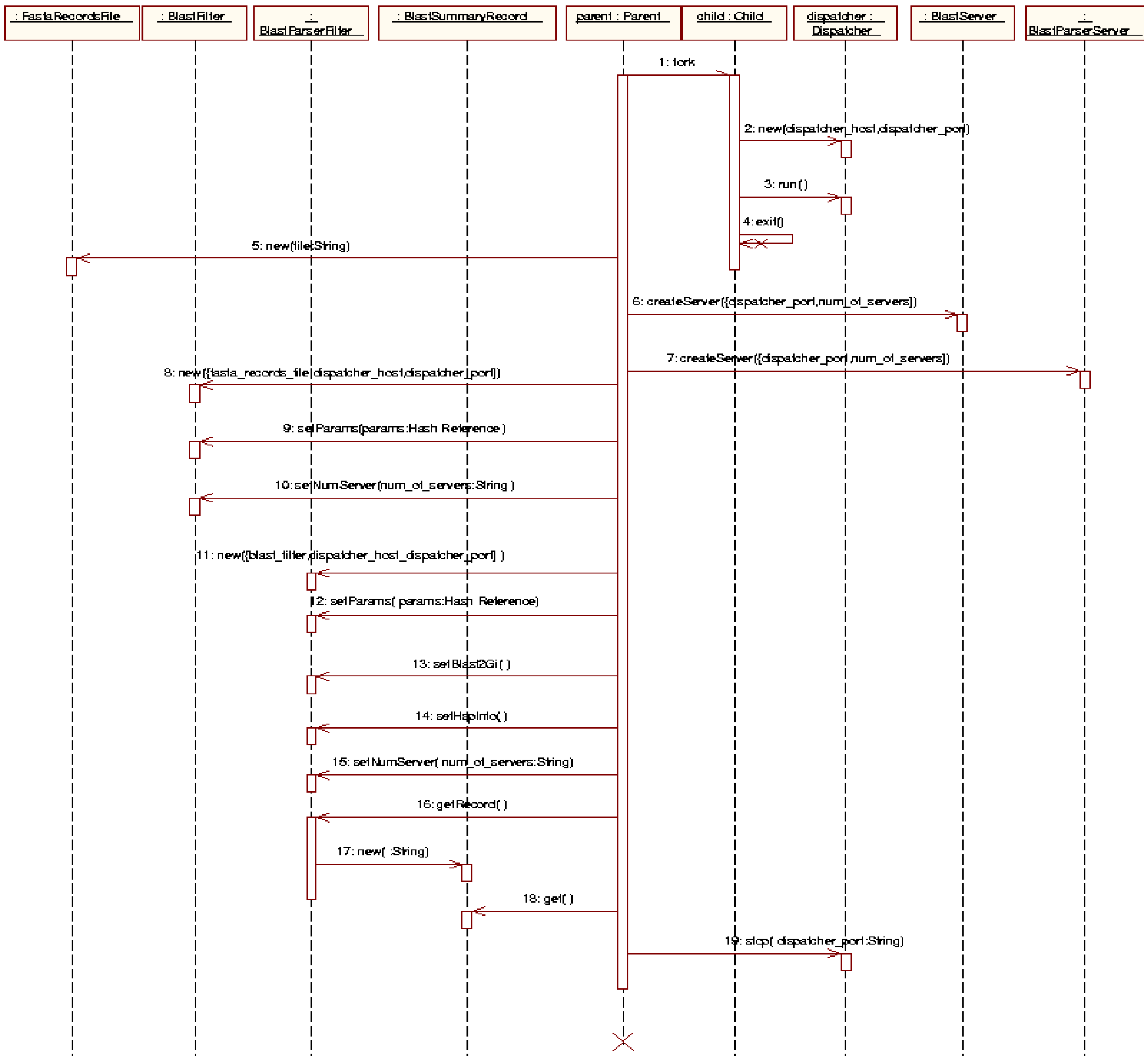
Architectural patterns, as used by us, specify the system-wide structural properties of an application and have an impact on the architecture of its subsystems. They provide a set of predefined subsystems, specify their

bioinformatics analysis involves processing data streams, this pattern was a natural choice. The Broker pattern is used to structure distributed software systems with decoupled components that interact by remote service invocations. Since our architecture aims to provide bioinformatics tools as services on the cluster, this pattern was also a natural choice.

Design patterns, as used by us, are medium-scale patterns that are smaller in scope than architectural patterns. A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly recurring structure of component communication that solves a general design problem within a particular context [14]. At the design level, our architecture utilizes the Pull Pipeline variant [14] of the Pipes and Filter architectural pattern and relies heavily on the Filter design pattern [13]. It also relies on the Client-Dispatcher-Server pattern [14] to implement the broker architecture.

3. Architecture

In genome analysis, input data sets can be thousands of sequences (DNA or amino acid) obtained either from an external source or as results from one of the tools in the pipeline. Tools generally process one element from the input data set at a time. For example PFAM [15] takes one query sequence from the input set of query sequences and searches it against a database of Hidden Markov models. The search result is independent of the results of other query sequences. This allows for splitting the input data set into mutually exclusive sets that individual servers can process.



: FastaRecordsFile
: BlastFilter
: BlastParserFilter
: BlastSummaryRecord
parent : Parent
child : Child
dispatcher : Dispatcher
: BlastServer
: BlastParserServer
1: fork
2: new(dispatcher_host,dispatcher_port)
3: run()
4: exit()
5: new(file:String)
6: createServer({dispatcher_port,num_of_servers})
7: createServer({dispatcher_port,num_of_servers})
8: new({fasta_records_file,dispatcher_host,dispatcher_port})
9: setParams(params:Hash Reference)
10: setNumServer(num_of_servers:String)
11: new({blast_filter,dispatcher_host_dispatcher_port})
12: setParams(params:Hash Reference)
13: setBlast2Gi()
14: setHspInfo()
15: setNumServer(num_of_servers:String)
16: getRecord()
17: new(:String)
18: get()
19: stop(dispatcher_port:String)

Fig. 2. Sequence diagram depicting the runtime behavior of the BioFilter architecture

Many bioinformatics tools search a query against a flat file database, BLAST is one of these tools. Attempts to increase the performance of these tools could split the database and distribute it on several nodes on the cluster. The reason for doing this is that if the database size is smaller, then it is more likely to fit into main memory avoiding disk access, and therefore the search is faster. However, this requires sending the same input element to many nodes. Synchronization issues, such as building the result for each input element by accumulating results from different nodes, affect tool performance, development time and cost.

Our approach to the database search relies on having a single monolithic database and process one input element at a time with a clear distinction between them (partial results don't have to be merged). We split the input data set into individual elements, and each of these elements goes to a different node for processing. The single monolithic database is present on a shared disk that is visible to each node. The alternative strategy of having the whole database on each node's local disk was examined but there was no significant performance improvement. However the local database could improve performance where the shared disk access is slow due to network bandwidth limitations.

There is another synchronization issue with the collection of results. When there is a one-to-one relationship between the input and output, the results should be collected and returned in the same order as the input is received. This does not slow down the process by a large degree as long as the deviation on the mean processing time for any individual job is small. In the case of genome analysis, we find this to be true the vast majority of the time, particularly if we pre-sort the input by the size of each input element. This is because the processing time for these tools is proportional to the input size. The results are collected from the nodes in the order in which individual jobs were launched and then passed to the next processing step in the pipeline or presented to the user if it is the last processing stage.

The static structure of the system is depicted using the UML class diagram (Figure.1). The classes represent an implementation that is based almost exclusively on the Filter design pattern and the Client-Dispatcher-Server design pattern. The runtime behavior of the system can be divided into three distinct phases. These phases are summarized using a blast filter and a blast parser filter in Figure 2 and examined in more detail in Figures 3-5. These phases are: i) *initialization* of the dispatcher and servers, ii) *transformation* of the original record by pulling it through a series of filters, and iii) *shut-down* of the dispatcher and servers.

In the initialization phase (Figure 2, Steps 1-7 and Figure 3), the parent process (running on a front-end node) forks off a single child process to run as the dispatcher. It then creates a new concrete source object that acts as the data source to the pipeline, in this case a FastaRecordsFile. Next, the required number of blast servers and blast parser servers to carry out the job are instantiated on backend nodes through the queuing system (in our case OpenPBS) as new processes. One of the first things a server does is to register with the dispatcher.

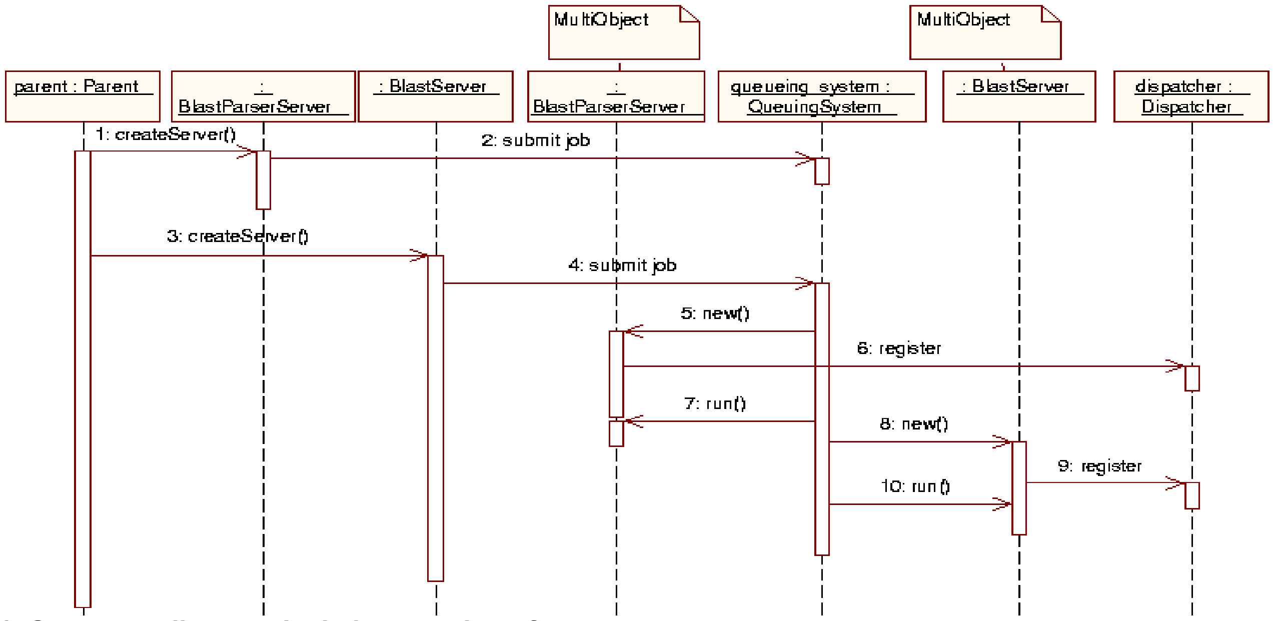


Fig. 3. Sequence diagram depicting creation of new servers

In the transformation phase (Figure 2, Step 8-18 and Figure 4), the parent instantiates the BlastFilter and the BlastParserFilter and sets the appropriate parameters on each filter. Filters are instantiated and remain active in the parent process. The filters pass work off to their respective servers that were launched in the initiation phase. A filter remembers how many servers it is interacting with in order to manage the number of records being transformed at any give time. The parent chains the filters together by providing a reference, for example, the BlastParserFilter is given a reference to a BlastFilter, thus chaining the two filters together. The getRecord method is called on the last filter in the pipeline, which in this example is the BlastParserFilter, and it returns a new BlastSummaryRecord. Figure 4 shows that the BlastSummaryRecord is the final product of a data stream being transformed by multiple filters and not just the BlastParserFilter.

The final phase represents a smooth shutdown of the system (Figure 5). In this phase, we shut down the dispatcher, which in turn shuts down the servers registered with it. Shutdown is initiated from the parent and occurs after all the input records have been processed.

Crash recovery is critical for distributed systems like BioFilter, where servers are distributed on different nodes and all of them are communicating with the dispatcher or the client filter on the parent node. Since the communication between these components is based on the TCP/IP socket framework, one of the components could go down and the interacting component would block for data on a socket causing the application to hang. A specific example is a pipeline with a single BlastFilter and 20 BlastServers, in which the filter takes 20 sequences at a time and sends them to the servers for processing and then blocks for results from these servers. If one of the nodes running a server goes down, the filter could block for results from that server forever. Timed sockets provide a reasonable solution to this problem. In the socket communication part of the system, the time that the component blocks for results is predetermined. This time limit may be variable for different components and for different communication combinations between components. In the filter implementations, the time limit is a settable parameter. As an example, a BlastFilter may block for 5 minutes for result of a data transformation on a backend server and then time out, however the same BlastFilter when communicating with the dispatcher may only block for 30 seconds and then time out. The variation in blocking time related to data transformation is dependent on computational intensity of the job submitted. The job is resubmitted to a backend server if the timeout occurs.

The input to output relationship gives rise to three kinds of Filter variants in the system: i) simple filters model a one-to-one relationship between input and output, ii) split filters model a one-to-many relationship between input and output, and iii) join filters model a many-to-one relationship between input and output.

Simple filters have a one-to-one mapping between input and output data. Fasta2TblFilter is an example of simple filter, which takes in a single FastaRecord and produces a single record in a tabulated format. Another example of a simple filter is a TranslationFilter, which takes in a single FastaRecord containing a nucleotide sequence and produces a single FastaRecord containing the translated protein sequence.

Split filters transform a single input data object into multiple output data objects, with an internal queue to hold the split results, providing a one to many mapping between input and output data. GlimmerFilter is an example of split filter that takes as input a single FastaRecord containing a nucleotide sequence (a chromosome or a microbial genome) and produces zero or more FastaRecords representing the genes contained in the input sequence. Another example of the split filter is an OrfFilter that takes a FastaRecord containing a single nucleotide sequence and produces zero or more FastaRecords representing the open reading frames of the input sequence.

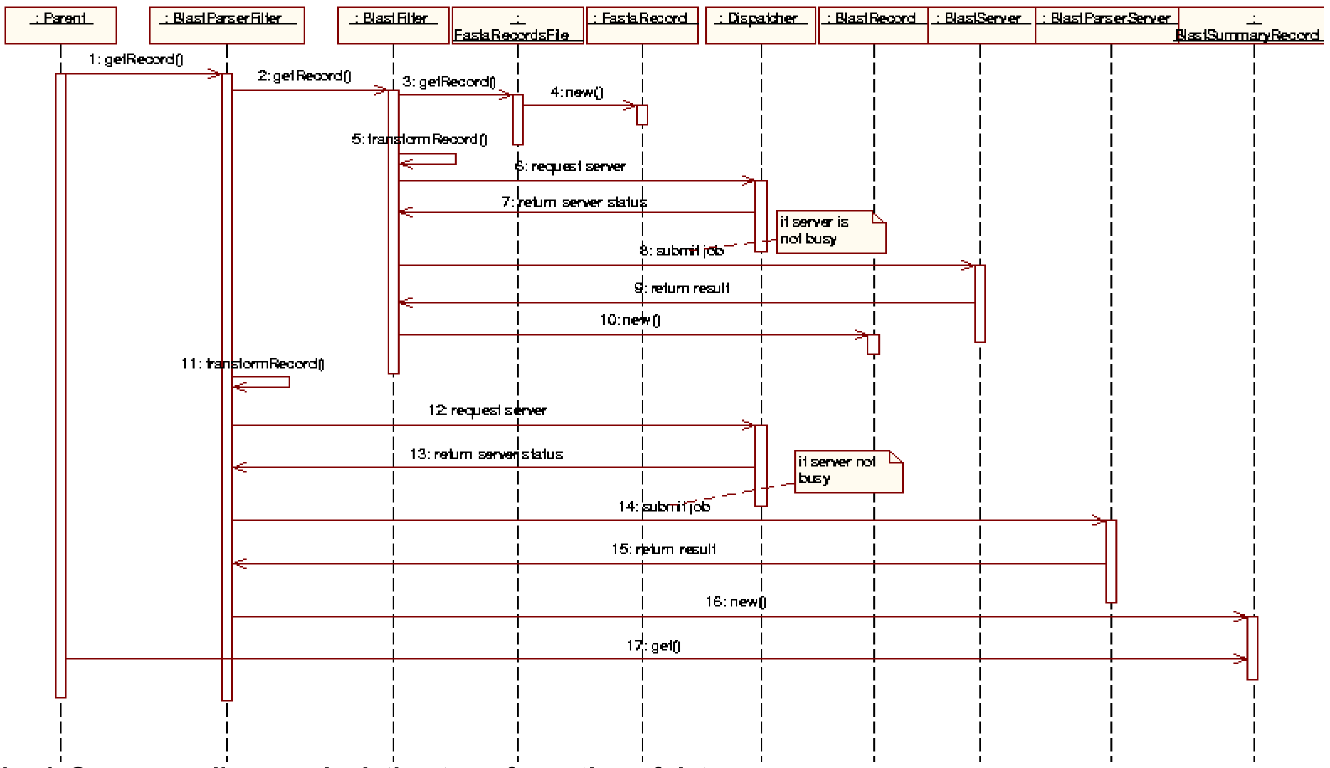


Fig. 4. Sequence diagram depicting transformation of data

Join Filters do the opposite of Split Filters. They collect all the input data objects and then transform them producing a single output data object. These filters provide a many to one mapping between input and output data. The BuildImmFilter (representing part of the GLIMMER software) is an example of join filter that takes in one or more FastaRecords in the tabulated format and produces a single Interpolative Markov model that can be used to predict genes.

In any dynamic real-world pipeline, there can be any possible combination of the above three variants as demonstrated in a gene prediction pipeline. In a gene prediction pipeline the concrete source is a FastaRecordsFile holding a single genome FastaRecord. This is input to the OrfFilter (split) that identifies open reading frames (ORF) in the genome and produces many Fasta formatted ORFRecords. Each of these ORFRecords is then passed to a Fasta2Tblfilter (simple), which just re-formats the ORFRecord into a tab-delimited format. All these tab-delimited formatted ORFs become input to the BuildImmFilter (join), which produces a single model file. This model file and the original genome FastaRecord from the concrete source are then fed to a GlimmerFilter (split) that produces nucleotide sequence records of the predicted genes in the genome. These records are then passed to a TranslationFilter (simple), which produces FastaRecords containing the corresponding amino acid sequences.

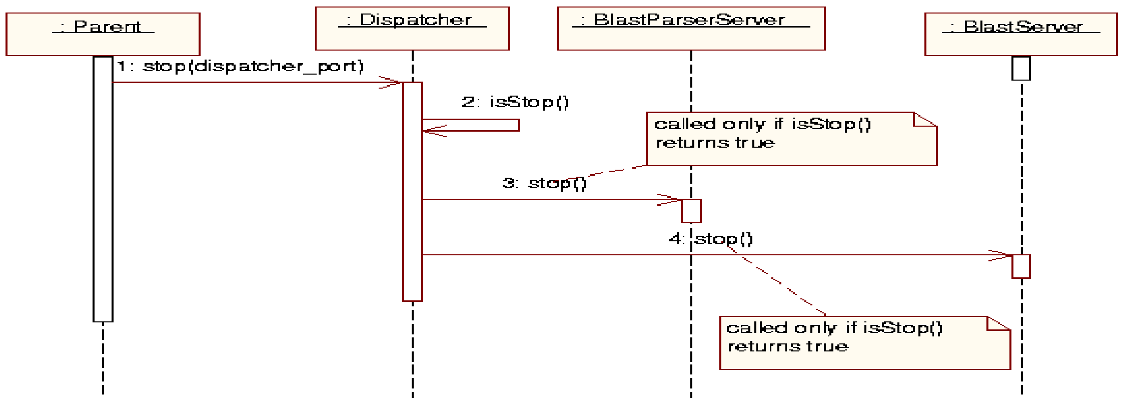


Fig. 5: Sequence diagram depicting shutdown of the system

At the present time, the architecture includes the OpenPBS queuing system on the cluster (http://www.openpbs.org). The design is flexible enough to incorporate new queuing systems and the implementation does not exclude the use of processors outside the cluster. The architecture supports interaction and job submission to servers on any remote machine, not necessarily one of the cluster nodes, and the implementation includes this functionality. This type of integration with nodes outside the cluster is possible since the communication is socket based and only requires that the server be ready to accept jobs. This flexibility has been useful in situations where the tool software is OS or hardware dependent and cannot be supported by the existing cluster hardware architecture or OS.

4. Performance results

4.1. System specifications

The Beowulf cluster has 240 nodes running Linux, each node with a single Pentium III, 1200 MHz processor, a 20 GB local disk space and memory ranging from 1GB-2GB. The nodes share a disk via the Netapps disk server (http://www.netapps.com). The network capacity of the channel from the node to the switch is 100 Mbps and of the channel from the switch to the Netapps is a 1 Gbps.

4.2. Benchmark tests

Benchmark tests were conducted on the cluster described above. A few of these tests are presented next. Many of our tests were examined using a calculation called speed-up ratio. The speed-up ratio for x nodes equals the job execution time on the initial number of nodes divided by the job execution time on x nodes.
For example.
Initial no. of nodes = 1; Job runtime on 1 node = 50 secs.
No. of nodes = 6; Job runtime on 6 nodes = 10 secs.
Speed-up Ratio (6 nodes) = 50/10 = 5.

Using 1000 identical bacterial protein sequences as queries against the NR database, the number of BLAST servers was varied from 1 to 50. As expected, the average time per BLAST search decreases as the number of nodes increase (Figure 6). Also, as the number of nodes increases the speed-up ratio increases (Figure 6). Also to determine that our architecture scales well as the input data size increases, we repeated the above test with 10000 sequences. The results for this test were almost identical to the one shown in Figure 6 thus proving that our architecture is data scalable.

We also carried out tests to demonstrate that the architecture is flexible enough to provide a performance speed-up for many bioinformatics tools and is not restricted to only speeding up BLAST. Three hundred identical bacterial protein sequences were used as input to the various tools like Blocks, Hmmpfam, and Prosite that had been plugged into the BioFilter architecture. For tRNAscan, the input was single record that represented 182,950 bases of a larger genome sequence. Performance gains for each of these tools were measured and are presented in Table 1 and 2. The other bioinformatics tools that have been deployed in our architecture to date and have realized similar performance gains are PSI-Blast, Prodom, Psort, Primer3 [16], and Phd [17]. Finally, speed-up is not the only reason that a tool is integrated into BioFilter. An example is the Glimmer program. This program runs sufficiently fast that speed-up is not an issue. This program and others like Coils and Seg have been included in the BioFilter architecture for the purpose of constructing pipelines that consist of many tools.

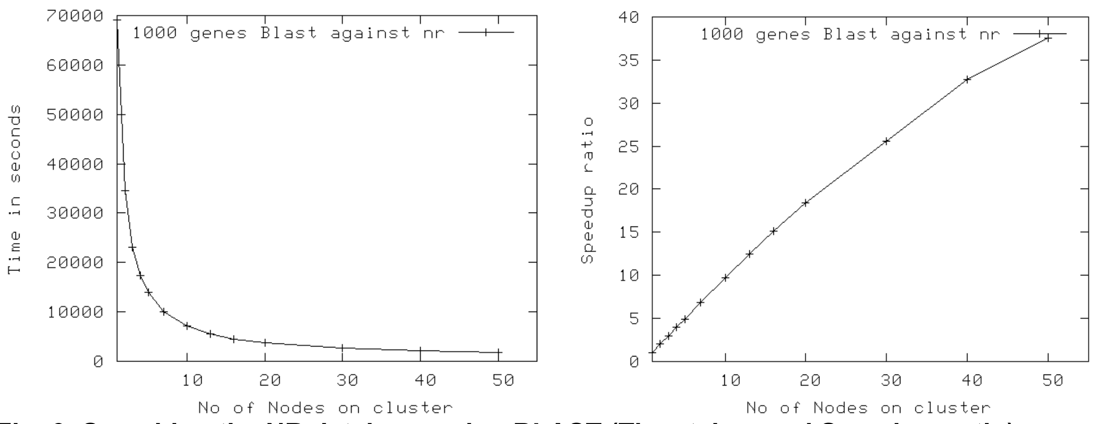


Fig. 6: Searching the NR database using BLAST (Time taken and Speed-up ratio)

Data in Table 1 and Table 2 show that for tools that are computationally light the speed-up ratio stops increasing and sometimes starts to decrease as the number of servers increase. This is because the total time to initialize and run the servers via the queuing system, distribute the data to be processed and collect the results is comparable to the time taken to actually process the data. In the tests we did, the speed-up ratio stops increasing at around 10-15 nodes for computationally light tools like Prosite, tRNAScan and Block whereas for computationally intensive tools like Blast and Hmmpfam it stops increasing around 40-50 nodes. However, if we were to use a tool that was more computationally intensive, then the number of nodes at which speed-up gain stops should be higher.

5. Future work

The process of incorporating a new tool into the architecture is straightforward. All the user has to do is write four derived classes; namely the ToolRecord, ToolRecordsFile, ToolFilter and ToolServer. Most of the functionality is part of the base classes that these four classes will extend. This makes the functionality of the derived classes lightweight and hence easy to program. After the initial design was laid out, it took us 3-4 man-hours for incorporating a new tool.

The process of incorporating a new tool does require a programmer with knowledge of Perl. We understand that a biologist/scientist might be interested more in the analysis of results than in writing these programs and we think that a GUI and/or a XML based language can reduce the step of writing code. Future work could focus on eliminating the step of writing code.

6. Conclusion

In this paper we have presented an architecture, that provides an environment for easy deployment of tools on the Beowulf cluster, that are embarrassingly parallel and whose input data set can be easily partitioned. The performance gain is two-fold, a development time performance gain is achieved since the plugging of a new stand-alone tool into BioFilter is very easy, and a runtime performance gain is achieved by parallel execution. The architecture is efficient yet simple due to synergy between simple and well-documented software patterns. It is also flexible in that it is independent of the cluster queuing system and has the capability to interact with nodes outside the cluster. The usage of Pipes and Filters pattern eliminates intermediate files, provides filter reuse and allows rapid prototyping of pipelines by filter recombination. The use of Client-Dispatcher-Server pattern provides server redundancy; location and migration transparency; reconfiguration of servers and

Table 1: Time taken for various tools versus the number of processors used (NT : Not Tested)

	Hmmpfam	Blocks	tRNAScan	Prosite
1	70924.97	4951.31	4191.27	1781.8
2	35551.28	2511.64	2120.17	1174.2
3	23734.37	1705.08	1442.82	805.40
4	17808.74	1307.40	1088.65	627.70
5	14505.87	1067.18	913.23	497.31
7	10267.57	796.24	687.70	391.10
10	7193.54	580.45	517.60	330.17
13	5763.63	493.28	408.84	250.74
16	4594.60	418.43	338.73	214.49
20	3707.27	429.14	298.26	248.93
30	2537.61	NT	NT	NT
40	2079.87	NT	NT	NT
50	1886.69	NT	NT	NT

Table 2: Speed-up Ratio for various tools versus the number of processors used (NT : Not Tested)

	Hmmpfam	Blocks	tRNAScan	Prosite
1	1	1	1	1
2	1.99	1.97	1.98	1.52
3	2.99	2.90	2.90	2.21
4	3.98	3.79	3.85	2.84
5	4.89	4.64	4.59	3.58
7	6.91	6.22	6.09	4.56
10	9.86	8.53	8.10	5.4
13	12.31	10.03	10.25	7.11
16	15.44	11.83	12.37	8.31
20	19.13	11.54	14.04	7.16
30	27.95	NT	NT	NT
40	34.10	NT	NT	NT
50	37.59	NT	NT	NT

fault tolerance. However, our architecture has problematic issues like difficult pipeline disaster recovery and error handling and high dependency on the interface of the dispatcher. In addition, the setup of pipelines can only be done by writing programs or scripts which use the existing classes in BioFilter. The solutions to these issues could be perceived as future work as could be the building of a GUI that allows construction of pipelines by non-programmers using simple drag-and-drop techniques.

Acknowledgements: This work was funded as part of the Sexually Transmitted Diseases Database project under an Inter Agency Agreement between the National Institute of Allergy and Infectious Diseases (NIAID/NIH) and the Department of Energy.

7. References

[1] S. Salzberg, A. Delcher, S. Kasif, and O. White.,"Microbial gene identification using interpolated Markov models" Nucleic Acids Research 26:2 (1998), 544-548.

[2] A.L. Delcher, D. Harmon, S. Kasif, O. White, and S.L. Salzberg., "Improved microbial gene identification with GLIMMER" Nucleic Acids Research, 27:23, 4636-4641.

[3] Lowe, T.M. & Eddy, S.R. (1997) ``tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence", Nucl. Acids Res., 25, 955-964.

[4] Altschul, S.F.,*et al.,* "Basic local alignment search tool.". J Mol Bio. 215 403-410(1990)

[5] Nakai K, Kanehisa M.,"Expert system for predicting protein localization sites in gram-negative bacteria." Proteins. 1991;11(2):95-110.

[6] Lupas A., Van Dyke M., and Stock J.,"Predicting Coiled Coils from Protein Sequences", Science 252:1162-1164.

[7]Wootton, J. C. and S. Federhen (1993)., "Statistics of local complexity in amino acid sequences and sequence databases.", Computers in Chemistry 17:149-163.

[8] Wootton, J. C. and S. Federhen (1996).," Analysis of compositionally biased regions in sequence databases. Methods in Enzymology 266: 554-571.

[9] Servant F, Bru C, Carrère S, Courcelle E, Gouzy J, Peyruc D, Kahn D (2002) ProDom: Automated clustering of homologous domains. Briefings in Bioinformatics. vol 3.

[10] J.G. Henikoff, E.A. Greene, S. Pietrokovski & S. Henikoff, "Increased coverage of protein families with the blocks database servers", Nucl. Acids Res. 28:228-230 (2000).

[11] S.Henikoff, J.G.Henikoff & S. Pietrokovski, "Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations", Bioinformatics (1999)

[12] Sigrist C.J., Cerutti L., Hulo N., Gattiker A., Falquet L., Pagni M., Bairoch A., Bucher P., "PROSITE: a documented database using patterns and profiles as motif descriptors." Brief Bioinform. 3:265-274(2002).

[13] Mark Grand. "Patterns in Java .Volume 1" ,Second Edition,Wiley Publication Inc ,2002

[14] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad,Michael Stal."Pattern Oriented Software Architecture Volume 1:A system of Patterns" Chicester,England:John Wiley and Sons, 1996.

[15] Bateman A, Birney E, Cerruti L, Durbin R, Etwiller L, Eddy SR, Griffiths-Jones S, Howe KL, Marshall M, Sonnhammer ELL. Nucleic Acids Res. 30:276-280 (2002)

[16] Steve Rozen, Helen J. Skaletsky (1996,1997,1998) Primer3.

[17] B Rost: PHD: predicting one-dimensional protein structure by profile based neural networks. Methods in Enzymology, 1996, 266, 525-539.

[18] Kai Hwang, Faye E Briggs ,"Computer architecture and parallel processing" McGraw Hill Inc 1984.