ANDY: A general, fault-tolerant tool for database searching on computer clusters

Supplementary Information

Andrew Smith[1,2,3,4], John-Marc Chandonia[2], and Steven E. Brenner[1,2]

Address for correspondence:
Steven E. Brenner
Department of Plant and Microbial Biology
461A Koshland Hall
University of California
Berkeley, CA 94720-3102
email: brenner@compbio.berkeley.edu

Affiliations:
1 - Department of Plant and Microbial Biology, University of California, Berkeley, CA 94720, USA
2 - Berkeley Structural Genomics Center, Physical Biosciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
3 - Department of Molecular Biophysics and Biochemistry and Department of Computer Science, Yale University, New Haven, CT 06520, USA
4 - Current Address: Department of Computer Science, Yale University, New Haven, CT 06520, USA

Introduction

This supplement provides a more detailed description of the ANDY infrastructure and features, reports results of several performance tests, and offers examples of configuration.

Infrastructure

The ANDY infrastructure consists of a server process, started by the user on the cluster head node, and clients, which the server submits to the DRM to be run on the compute nodes, as shown in Figure S-1. Use of the cluster DRM allows ANDY clients to intersperse fairly with jobs submitted by other users, according to the local policy for allocation of cluster resources. When each ANDY client process is started by the DRM on a compute node, it contacts the server process to request configuration information for the run (command template, environment variables to set, what results to relay, etc.). Clients then enter a loop in which they repeatedly request tasks from the server, interpolate the command template with values specific to the task, execute the task, and send results and notification of errors back to the server. For example, if the entire job were to compare a large database of sequences to another, a single task might involve comparing a small number of sequences from one database against the other database. The granularity of each task may be specified by the user.

ANDY may be run in *dedicated mode*, in which a specified number of client processes are submitted and these continue to execute tasks until all are completed, or in *fair mode*, in which each client exits after performing a fixed number of tasks. In the latter mode, the server continues to create new ANDY clients and submit them to the queue as required until all tasks are completed. By making clients take a relatively short period of time on each node, other jobs have a chance to intersperse with ANDY clients in the queue.

Performance

A key improvement of ANDY over similar tools is support for input, output, and inter-command communication through named pipes, in addition to files and unnamed Unix pipes. Pipes allow information to be passed in memory between consecutive steps in a pipeline of programs being run, rather than being written to disk. This gives a significant performance advantage, especially for larger jobs running on a busy cluster. Figure S-2 shows performance at maximal CPU usage for a BLAST run (64 jobs running on a 32-node 64-CPU cluster managed by the PBSPro DRM), showing how ANDY performance scales with the search database size. Note the large advantage of named pipes over files in this example. Note also how scalability improves as database size increases, as is expected due to a lower proportion of overhead for larger task sizes. It is likely that the cause of the performance disadvantage for files is caused by competition for the single disk on each node, as our DRM schedules consecutive jobs on the same node.

Use of named pipes rather than files to store temporary results before summarization allows ANDY to achieve nearly linear scaling in performance for a range of CPUs that is typical for modern clusters. Figure S-3 shows performance for varying numbers of CPUs on a BLAST run of approximately 10,000 query sequences against a medium size database, on the same cluster managed by the PBSPro DRM. While the use of keepalive sockets gives some advantage, especially for file operation, most striking is the large advantage of named pipes over files. In this example, named pipes provide double the performance of files. For reasonably sized databases, ANDY CPU efficiency approaches 90% when using named pipes.

Figure S-4 shows results of the same performance test as Figure S-3, on a second cluster managed by the Sun GridEngine DRM. This cluster also has 32 dual-processor nodes. Keepalive sockets were not used in this test. As this cluster was optimized to avoid competition for disk resources (the DRM was configured to schedule consecutive jobs on separate nodes), the performance advantages of named pipes over files was smaller than for the PBSPro cluster. Except for one data point, ANDY CPU efficiency remained above 90% throughout the test when using named pipes.

ANDY provides two implementations of named pipes: native and memory-buffered. In the former case (for which results are described above), Unix named pipes are simply substituted for files; the ANDY client manages their creation and cleanup, and interpolates the full path names into the command lines of tasks being executed. Although fast, this type of named pipe has several disadvantages: it cannot be randomly accessed and the contents can only be read once. To solve the latter problem, ANDY also provides memory-buffered pipes, in which one named pipe is the data source, a separate buffering process reads from it, buffering in memory as necessary, and the data is written to one or more other named pipes. Although memory-buffered pipes still cannot be randomly accessed, they provide a generalized "multi" tee, allowing the same data to be efficiently distributed to several tasks without being written to disk. Unix named pipes have a fixed buffer size which can lead to bottlenecks where one process must wait to write to a buffer until another process reads from it; in the worst case, this can create a deadlock where two processes each wait for the other indefinitely. Memory-buffered pipes can alleviate this situation by creating pipes effectively limited only by a machine's memory. As use of memory-buffered pipes incurs a performance penalty relative to named pipes, native pipes are preferred in cases where the additional flexibility is not required.

Configuration

Users configure ANDY through an XML configuration file that specifies a parameterized command template in common Unix shell syntax, along with the locations and types (e.g., FASTA file) of data sources which provide values for each parameter. An example showing portions of a XML configuration file is given in Figure S-5. In this example, sequences in the files `test1.fa` and `test2.fa` are compared to `astral1.65.fa`, a complete set of sequences from the ASTRAL database (Chandonia, et al., 2004) using PSI-BLAST (the `blastpgp` program). ANDY breaks the job into individual tasks containing a small number of sequences. This example uses memory buffers to send the output of the PSI-BLAST jobs to both a client-side summarizer and to gzip. To instantiate each task on a node, ANDY interpolates values specific to the task into variables (words surrounded by __ symbols) in the command line template, then executes the task. ANDY loops over all data sources for each variable parameter until the entire job has been broken into tasks. Values for each variable parameter may be enumerated directly in the configuration file, or read from several types of data sources, including FASTA files and directory listings. Variable enumeration is particularly useful for comparing the results of changing one or more parameters in the command template (e.g., values for the -h or -j parameters).

References cited in the Online Supplement

Bateman, A., Coin, L., Durbin, R., Finn, R.D., Hollich, V., Griffiths-Jones, S., Khanna, A., Marshall, M., Moxon, S., Sonnhammer, E.L., Studholme, D.J., Yeats, C. and Eddy, S.R. (2004) The Pfam protein families database, *Nucleic Acids Res*, **32 Database issue**, D138-141.

Chandonia, J.M., Hon, G., Walker, N.S., Lo Conte, L., Koehl, P., Levitt, M. and Brenner, S.E. (2004) The ASTRAL Compendium in 2004, *Nucleic Acids Res*, **32 Database issue**, D189-192.

Wunderlich, Z., Acton, T.B., Liu, J., Kornhaber, G., Everett, J., Carter, P., Lan, N., Echols, N., Gerstein, M., Rost, B. and Montelione, G.T. (2004) The protein target list of the Northeast Structural Genomics Consortium, *Proteins*, **56**, 181-187.

Figure Legends

Figure S-1. Overview of ANDY infrastructure. ANDY components are shown in orange. The server process submits jobs, each containing one client process, through the DRM (light blue), where they may intersperse with other users' jobs (purple). Upon being started, clients communicate with the server (green arrows) to exchange configuration data and receive tasks. Clients start other programs (yellow) as specified in the command template to perform each task. Information is exchanged (red arrows) between these programs and the ANDY client though named pipes, memory buffers, or files. One or more sets of results from each client may be sent back to the server (blue arrows), where they are saved to disk or optionally subjected to additional processing (yellow, on head node).

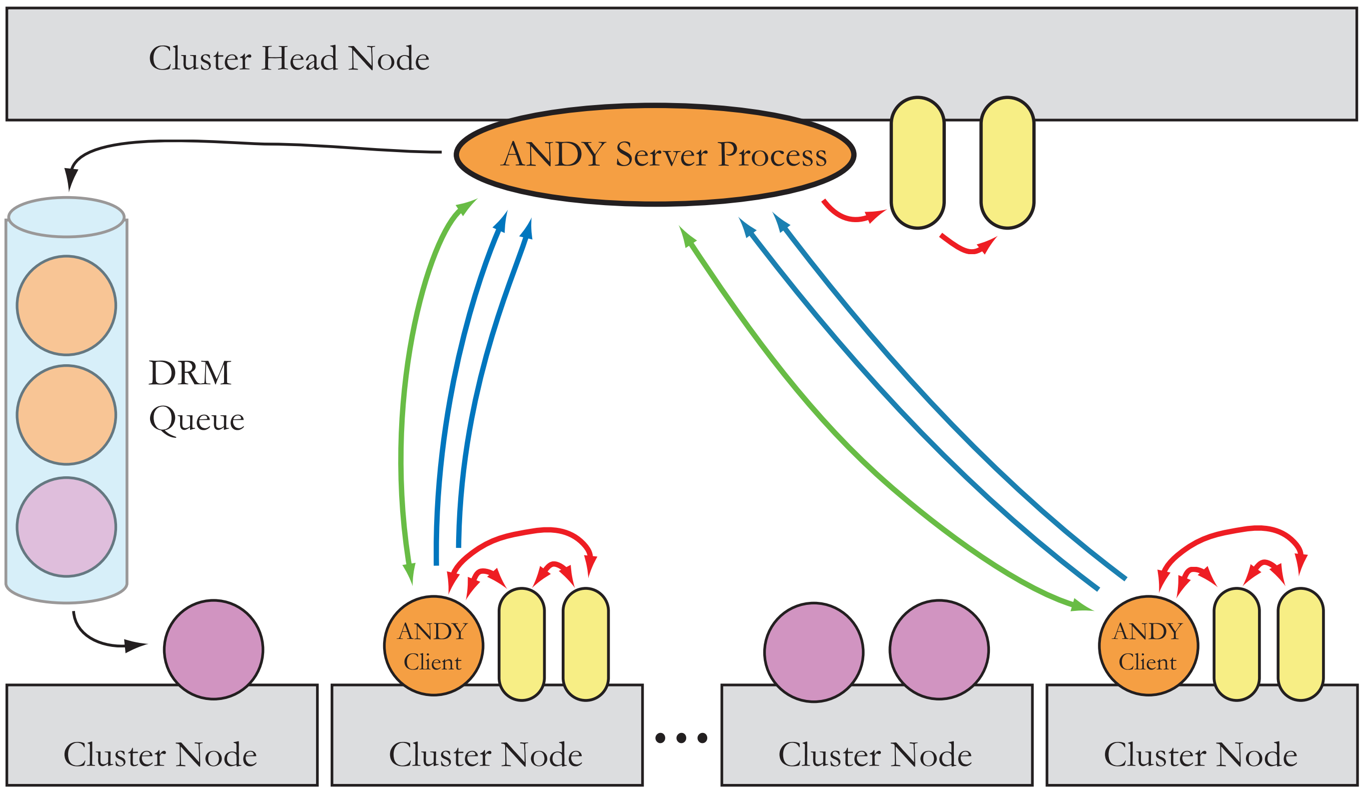
Figure S-2. Performance and scalability of ANDY. Results compare PIPE versus FILE, client/server keepalive sockets versus not, and increasing BLAST database size (i.e., number of database sequences). The query database is a set of all protein targets from the Northeast Structural Genomics Consortium (Wunderlich, et al., 2004) in Nov 2004, about 10000 sequences. The search databases were various sized subsets of sequences from the Pfam-A database of protein families (Bateman, et al., 2004). Our cluster has 32 dual-processor nodes (64 CPUs), where each node's 2 CPUs share a common memory and local disk, so this shows scalability at maximum cluster utilization. The cluster was managed by the PBSPro DRM. CPU efficiency is the percentage of theoretically possible linear speedup achieved on multiple CPUs versus the CPU time required to run a job on a single CPU. The dip in the green line ("PIPE, keepalive") is a small aberration, likely caused by transient issues on the cluster during the long run to obtain the data, but the trend is clear and unambiguous. For simplicity the single CPU baseline time was measured using the ANDY tool. As a control, we also did such a single CPU run on a quiescent cluster node without ANDY or the DRM for BLAST database size of 531384 sequences, and found a negligible time difference (4880 sec for this single CPU run versus about 4900 to 4950 for the equivalent run with ANDY via the cluster).

Figure S-3. CPU efficiency for varying numbers of CPUs for a BLAST run with a fixed size search database. The query database is a set of all protein targets from the Northeast Structural Genomics Consortium in Nov 2004, about 10,000 sequences. The search database was a subset of sequences from the Pfam-A database of protein families (Bateman, et al., 2004), containing 531,384 sequences. Our cluster has 32 dual-processor nodes (64 CPUs), where each node's 2 CPUs share a common memory and local disk. The cluster was managed by the PBSPro DRM. CPU efficiency is the percentage of theoretically possible linear speedup achieved on multiple CPUs versus the CPU time required to run a job on a single CPU. It is likely that the cause of the performance disadvantage for files is caused by competition for the single disk on each node, as our DRM schedules consecutive jobs on the same node. Average performance for jobs running on an even number of CPUs is lower than for jobs running on an odd number of CPUs, and this phenomenon is particularly apparent when jobs are running on a small number of CPUs (<10). Although the DRM may be optimized to avoid competition for resources, use of PIPEs rather than FILEs for inter-process communication gives a significant performance advantage in cases where such competition is unavoidable (i.e., a busy cluster).

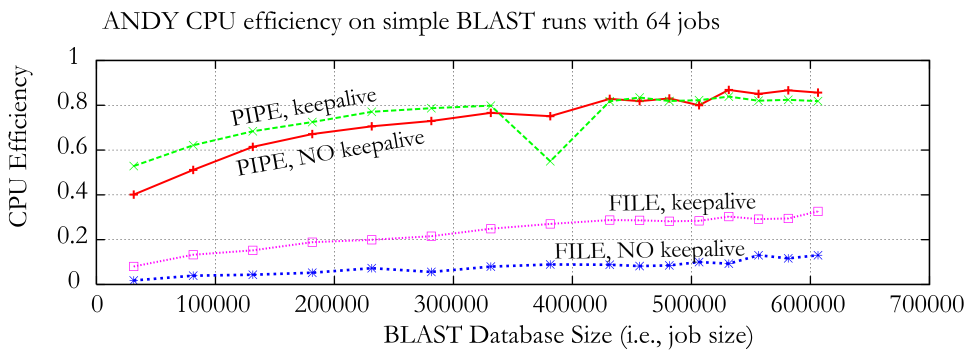
Figure S-4. CPU efficiency for varying numbers of CPUs for a BLAST run with a fixed size search database, using the GridEngine DRM. The test was identical to that from Figure S-3, but run on a second test cluster managed by the GridEngine DRM. This cluster also has 32 dual-processor nodes (64 CPUs), where each node's 2 CPUs share a common memory and local disk. The GridEngine DRM was configured to not schedule consecutive jobs on the same node. As expected, this provides relatively better scaling in the FILE test than our PBSPro cluster, although PIPEs provide better performance than FILEs over most of the range of CPUs tested.

Figure S-5. Example of parameterized command templates for ANDY. The '&' character separates commands which will run in parallel. The "&&&" symbol separates commands which run on the client nodes from those which run on the server. The command template runs PSI-BLAST (blastpgp) on a sequence called by a variable name, __SEQ__. The output is sent to a variable named __OUT__. The configuration file specifies that __SEQ__ will be a named pipe containing sequences read consecutively from two FASTA files: test1.fa and test2.fa. Creation and cleanup of this randomly named pipe are handled by the ANDY client. Each client task will handle one or more sequences, as specified by the user (not shown). The __OUT__ variable will be a memory buffer, which is also handled by the ANDY client. In this example, a local summarizer (sumClient) runs on each client node, extracting items of interest (e.g., E-values and raw scores) from the BLAST output into a summary report. A server-side summarizer (sumServer) amalgamates summary reports from each client into a global summary named "global-summary.txt". As the client-side summarizer runs, the raw output is also compressed using gzip and stored in a file. This compressed output will be written to a file and stored on the server, named according to the index of the first sequence processed in the task. Full documentation of the file format is available on the ANDY website.

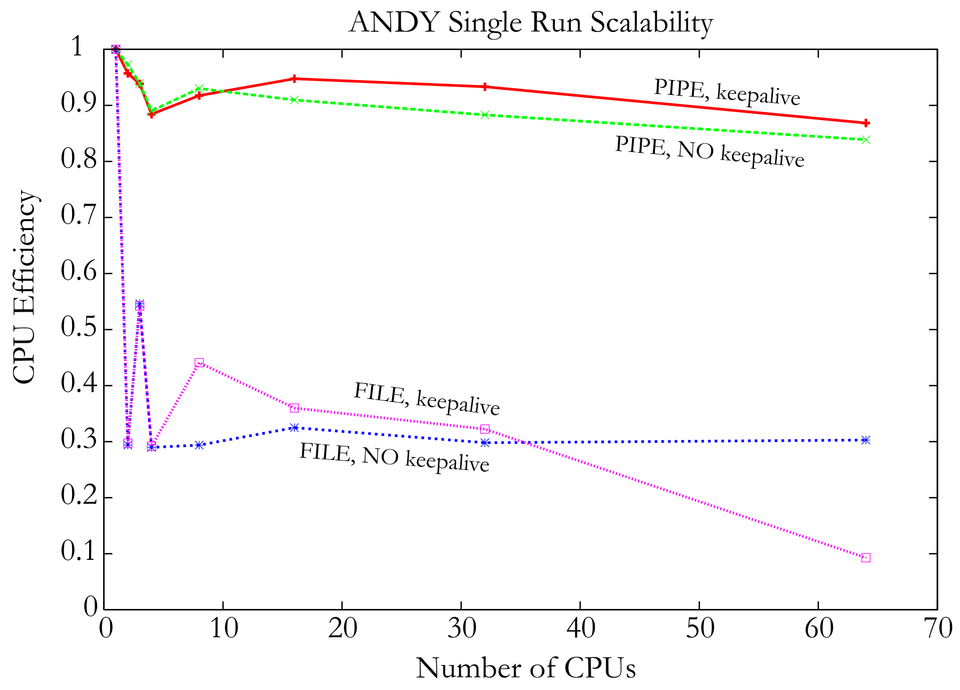
Smith, Chandonia, & Brenner, Figure S-1



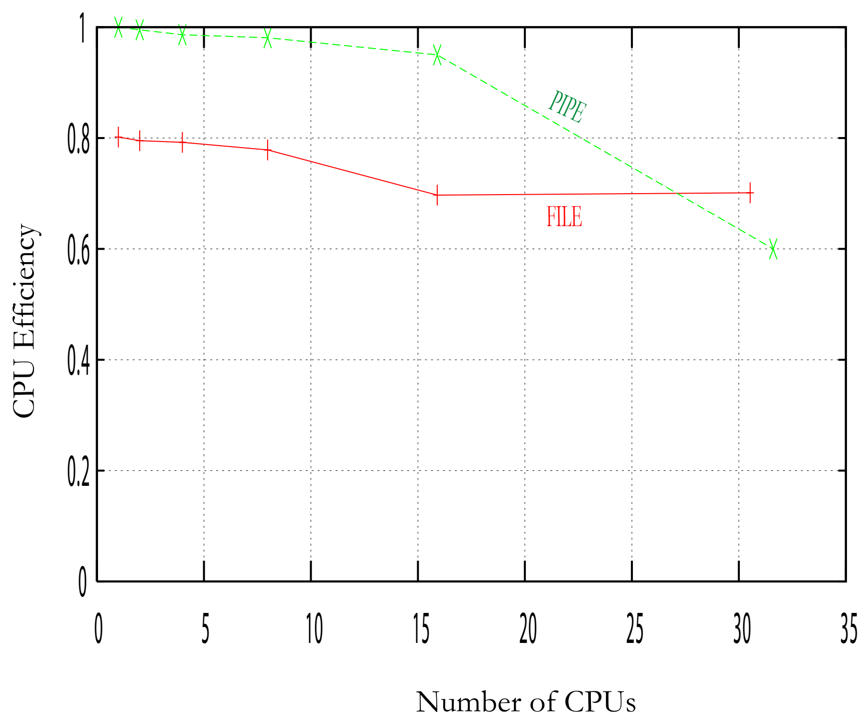
Smith, Chandonia, & Brenner, Figure S-2



Smith, Chandonia, & Brenner, Figure S-3



Smith, Chandonia, & Brenner, Figure S-4



Smith, Chandonia, & Brenner, Fig S-5:

Command template:

```
blastpgp -i __SEQ__ -d astral-1.65.fa -j 6 -h 0.001 -o __OUT__ &
sumClient -i __OUT__ -o __SUM_OUT__ &
gzip -9 < __OUT__ > __ZIP_OUT__ &&&
sumServer -i __SUM_OUT__ -o global-summary.txt
```

Variable configuration:

```
<VARIABLE NAME="__SEQ__" TYPE="PIPE">
  <DATA_SOURCE TYPE="FASTA_FILE">
    <LIST>
     <ITEM>/home/db/test1.fa</ITEM>
     <ITEM>/home/db/test2.fa</ITEM>
    </LIST>
  </DATA_SOURCE>
</VARIABLE>

<VARIABLE NAME="__OUT__" TYPE="MEM_BUF"/>

<VARIABLE NAME="__ZIP_OUT__" TYPE="FILE">
  <SERVER_SAVE>[__SEQ__.INDEX1].bl.gz</SERVER_SAVE>
</VARIABLE>
```