BLAST Training

Daniel Udwary
NERSC Data Science Engagement Group
December 16, 2015
Why am I running a BLAST tutorial?

BLAST makes up a very significant portion of the JGI’s compute workload.

And, we’ve found that most BLAST usage on Genepool could be done more efficiently…
Simple dotplots
## How it works

Compare my name:  
Daniel Wayne Udwary  

With my father's:  
Wayne John Udwary

<table>
<thead>
<tr>
<th>D</th>
<th>A</th>
<th>N</th>
<th>I</th>
<th>E</th>
<th>L</th>
<th>W</th>
<th>A</th>
<th>Y</th>
<th>N</th>
<th>E</th>
<th>U</th>
<th>D</th>
<th>W</th>
<th>A</th>
<th>R</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>A</td>
<td>Y</td>
<td>N</td>
<td>E</td>
<td>J</td>
<td>O</td>
<td>H</td>
<td>N</td>
<td>U</td>
<td>D</td>
<td>W</td>
<td>A</td>
<td>R</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### How it works

Compare my name:
Daniel Wayne Udwary

With my wife’s:
Megan Eileen Welsh

<table>
<thead>
<tr>
<th>D</th>
<th>A</th>
<th>N</th>
<th>I</th>
<th>E</th>
<th>L</th>
<th>W</th>
<th>A</th>
<th>Y</th>
<th>N</th>
<th>E</th>
<th>U</th>
<th>D</th>
<th>W</th>
<th>A</th>
<th>R</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Number of matches: 5
Local vs Global Alignment

Which is “best”?

BLAST searches for the best local alignment(s)
“Basic Local Alignment Search Tool”
Recommendation Zero:

• Is BLAST the best tool for the job?
• Which BLAST executable is going to give you the results you actually want?

- blastn
- blastp
- blastx
- tblastn
- tblastx
- deltablast
- legacy_blast.pl
- psiblast
- rpsblast
- rpstblastn

**Computation time:**
tblastx >> blastx > blastp > blastn

**Possible alternatives:**
LAST
BLAT
MegaBLAST
hmmer
usearch
short-read aligners
phylogenetic tools
The BLAST algorithm

• **Four major steps:**
  – Seeding
    • Drawing the dots
  – Extension
    • Drawing the lines
  – Evaluation
    • Figuring out which lines are best
  – Output
    • Writing the results out to disk
Seeding

- Significant alignments have “words” in common.

![Example of word alignment]

When comparing two sequences, BLAST breaks sequences into each word
- Each word and its location stored in a database prior to execution of the search.
- So, word search/matching is just database lookup. (Very fast)

Problems (depending on word size)
-- But some words too common to be “important”
-- Some related sequences may have no common words

So, BLAST employs a “neighborhood” of similar words to generate “word hits” if score $T$ threshold is reached.
Next, generate locations of “word hits” between the two sequences.

Can adjust word size and $T$ threshold to balance sensitivity with speed.

Higher $T$ = fewer word hits = faster algorithm

Protein sequence alignments:
-- Generally word size = 2 or 3
-- For speed, often $T$ is set arbitrarily large
-- Filtering of “low-complexity” regions (ie repetitive sequences)

Some differences when aligning nucleotides:
-- Use only exact-match words (ie no $T$)
-- Adjust sensitivity with word size
**Extension**

Word hits are extended to find additional matching regions

The quick brown fox jumps over the lazy dog.  
The quiet brown cat purrs when she sees him.

Assume ‘T’ in ‘The’ is the seed, and we extend to the right.

First mismatch. Do we continue or not?

Create variable $X$, that represents how much the score is allowed to drop off since the last maximum. So, for $X=5$:

The quick brown fox jump  
The quiet brown cat purr  
123 45654 56789 876 5654 \(-\) score  
000 00012 10000 123 4345 \(-\) drop off score

Note: Originally, gaps not considered.  
Modern implementations use scoring systems with gap penalties considered.
**Evaluation**

Determine if alignment(s) are statistically significant
- HSP = High-scoring Segment Pair

Goal: Set an alignment threshold score to eliminate small HSPs that may not be relevant.

Common thresholds:
- Certain number of top hits (default = 500)
- % identity (default = not used)
- Evaluate (default = 10)

Alignments must also be “consistent”

ie always “downhill”
BLAST output

Standard format
- human readable, but not very easy to parse

-outfmt <String>
alignment view options:
  0 = pairwise,
  1 = query-anchored showing identities,
  2 = query-anchored no identities,
  3 = flat query-anchored, show identities,
  4 = flat query-anchored, no identities,
  5 = XML Blast output,
  6 = tabular,
  7 = tabular with comment lines,
  8 = Text ASN.1,
  9 = Binary ASN.1,
  10 = Comma-separated values,
  11 = BLAST archive format (ASN.1),
  12 = JSON Seqalign output,
  13 = JSON Blast output,
  14 = XML2 Blast output
The *modern* BLAST algorithm

- **Four major steps:**
  - **Seeding**
    - Break sequence(s) into words, search for similar words in database
    - Calculate E-value to select HSPs
    - Pass selected number of HSPs to extension step
  - **Extension**
    - Generate alignment
  - **Evaluation**
    - Perform additional evaluations (%identity, %positives, %gaps)
  - **Output**
    - Write out to disk in one of 16 formats
Recommendation 1:

• Use a modern BLAST
  – “module load blast+” = most recent NCBI BLAST package
    • v2.2.31+
  – “module load blast” = “legacy” blast, deprecated in 2007
    • v2.2.26
  – Different executables, slightly different outputs
    • So, we keep the old one around for older software/pipelines
    • Don’t use legacy blast, if you can at all help it!!!

  – BLAST+ also shows much better parallelization
    • (much more on that in a few slides!)
Recommendations 2 & 3

• **Use a realistic E-value threshold**
  – “-evalue
  – Know what it is you want to know from your results.
  – In large databases (nt, nr) anything above 1E-5 is firmly in the “Twilight Zone” of sequence homology.
    • Default of 10 is just asking for lots of extra compute time and meaningless results
  – For annotation, 1E-20 probably better for interpreting functional relationships

• **Set a sequence output limit**
  – “-max_target_seqs 10” for top ten hits. Top 5?
  – Does anyone ever want 500 results?

(There may need to be a balance between these two settings...)
A quick thought on Genepool and parallelization...

I need to run 1000 blast searches. So, I’ll just set up a task array of 1000 jobs, and it will go a lot faster!

Then, imagine this picture with ~1000 bowls, and 140,000 puppies per day.
Putting parallelization to the test

Annotation simulation
• Task: use blastp to compare 100 “putative” protein sequences against NCBI refseq_protein database
• First 100 protein sequences from *Streptomyces coelicolor*

  2.2.26 “Legacy” blastall –p blastp
  vs
  2.2.31 “Blast+” blastp

Contaminant detection simulation
• Task: use blastn to compare 100 1kb “contig” sequences from a bacterial plasmid against NCBI nt database
• 1kb sequences from bacterial plasmid

  2.2.26 “Legacy” blastall –p blastn
  vs
  2.2.31 “Blast+” blastn
## “Annotation” test

<table>
<thead>
<tr>
<th>Executable</th>
<th>input</th>
<th>threads/job</th>
<th>Evaluate</th>
<th>output sequences</th>
<th>time to results (not counting queue time)</th>
<th>min/max walltime per job</th>
<th>total walltime</th>
<th>walltime hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.26</td>
<td>1x100</td>
<td>1</td>
<td>10</td>
<td>500/250</td>
<td>28 m</td>
<td>95/1599</td>
<td>81034</td>
<td>22.5</td>
</tr>
<tr>
<td>2.2.26</td>
<td>1x100</td>
<td>1</td>
<td>1.00E-20</td>
<td>500/250</td>
<td>37 m</td>
<td>218/2195</td>
<td>71806</td>
<td>19.9</td>
</tr>
<tr>
<td>2.2.26</td>
<td>1x100</td>
<td>1</td>
<td>1.00E-20</td>
<td>5/5</td>
<td>34 m</td>
<td>195/2053</td>
<td>59674</td>
<td>16.6</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>1x100</td>
<td>1</td>
<td>1.00E-20</td>
<td>5</td>
<td>26 m</td>
<td>208/1558</td>
<td>59734</td>
<td>16.6</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>1</td>
<td>0.1</td>
<td>500</td>
<td>595 m</td>
<td></td>
<td>35731</td>
<td>9.9</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>8</td>
<td>0.1</td>
<td>500</td>
<td>90 m</td>
<td></td>
<td>5364</td>
<td>1.5</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>16</td>
<td>0.1</td>
<td>500</td>
<td>84 m</td>
<td></td>
<td>5061</td>
<td>1.4</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>16</td>
<td>1.00E-20</td>
<td>500</td>
<td>36 m</td>
<td></td>
<td>2164</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Putting parallelization to the test

“Contaminant detection” test

100 sequences blasted vs NCBI “nt” database with default parameters, except specification of number of threads

<table>
<thead>
<tr>
<th>Executable</th>
<th>input</th>
<th>threads/job</th>
<th>time to results (not counting initial queue time)</th>
<th>total walltime</th>
<th>walltime minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.26</td>
<td>1x100</td>
<td>1</td>
<td>55 min</td>
<td>9901.5</td>
<td>165.0</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>1x100</td>
<td>1</td>
<td>16 min</td>
<td>4123.3</td>
<td>68.7</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>1</td>
<td>3 min</td>
<td>165.2</td>
<td>2.8</td>
</tr>
<tr>
<td>2.2.31+</td>
<td>100x1</td>
<td>8</td>
<td>1 min</td>
<td>62.3</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Why so much improvement with multiple inputs?
Recommendations 4 & 5:

• **Batch your BLAST input!**
  
  – Current BLAST treats multiple input sequences as a single one for seeding purposes, then breaks up the results before extension
  
  – With many sequences, this dramatically reduces the number of database queries

• **When batching many sequences, use “–num_threads 8”**
  
  – Current version doesn’t seem to scale above 8 threads
How is JGI doing with all that currently?

Mostly outdated versions of blast are being used

- 1104031 /global/dna/projectdirs/microbial/omics-pamm/lib/BLAST/blastall (2.2.21)
- 698394 blastall (probably 2.2.26)
- 653252 blastx
- 600898 /global/dna/projectdirs/fungal/pipeline/FRK/bin/euk/linux/blast/blastall (2.2.4)
- 274490 /global/dna/projectdirs/fungal/pipeline/2015-01-09_v1.9.1/bin/euk/linux/blast/blastall (2.2.4)
- 183805 /global/dna/projectdirs/fungal/pipeline/FRK/utils/iprscan//bin/binaries/blast/blastall (2.2.6)
- 160761 /usr/common/jgi/aligners/blast/2.2.26/bin/blastall (2.2.26)
- 138106 /global/dna/projectdirs/microbial/omics-pamm/lib/rpsblast/rpsblast (2.2.23)
- 46672 bin/blast/2.2.6/blastall (2.2.6)
- 24411 blastn
- 16832 bin/blast/2.2.19/blastall (2.2.19)
- 15190 megablast
- 13789 /projectb/sandbox/plant/iprscan/iprscan.uge/bin/binaries/blast/blastall (2.2.19)
- 12505 /usr/common/jgi/aligners/blast+/2.2.28/bin/blastn
- 10769
- 7926 /usr/common/jgi/aligners/wublast/20060510/blastp
- 5101 /global/dna/projectdirs/plant/tools/compgen/rmblast/DEFAULT/bin/rmblastn
- 2428 /usr/common/jgi/aligners/blast+/2.2.29/bin/blastn
- 2415 /usr/common/jgi/aligners/blast+/2.2.31/bin/blastn
- 1717 /global/projectb/sandbox/IMG/img/dataLoad5/tools/blast/rpsblast
- 1283 /usr/common/jgi/aligners/blast+/2.2.26/bin/tblastn
How is JGI doing with all that currently?

Many users using very large E-values

<table>
<thead>
<tr>
<th>count</th>
<th>evalue threshold</th>
<th>count</th>
<th>evalue threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>1719</td>
<td>1.00E-100</td>
<td>770353</td>
<td>1.00E-01</td>
</tr>
<tr>
<td>2485</td>
<td>1.00E-50</td>
<td>769204</td>
<td>1.00E-05</td>
</tr>
<tr>
<td>595500</td>
<td>1.00E-30</td>
<td>743675</td>
<td>1.00E-03</td>
</tr>
<tr>
<td>41033</td>
<td>1.00E-20</td>
<td>692386</td>
<td>1</td>
</tr>
<tr>
<td>13601</td>
<td>1.00E-10</td>
<td>595500</td>
<td>1.00E-30</td>
</tr>
<tr>
<td>4620</td>
<td>1.00E-08</td>
<td>165291</td>
<td>1.00E-02</td>
</tr>
<tr>
<td>769204</td>
<td>1.00E-05</td>
<td>121467</td>
<td>1.00E-04</td>
</tr>
<tr>
<td>121467</td>
<td>1.00E-04</td>
<td>56398</td>
<td>1.00E+01</td>
</tr>
<tr>
<td>743675</td>
<td>1.00E-03</td>
<td>41033</td>
<td>1.00E-20</td>
</tr>
<tr>
<td>165291</td>
<td>1.00E-02</td>
<td>13601</td>
<td>1.00E-10</td>
</tr>
<tr>
<td>770353</td>
<td>1.00E-01</td>
<td>4620</td>
<td>1.00E-08</td>
</tr>
<tr>
<td>692386</td>
<td>1</td>
<td>2485</td>
<td>1.00E-50</td>
</tr>
<tr>
<td>56398</td>
<td>1.00E+01</td>
<td>1719</td>
<td>1.00E-100</td>
</tr>
<tr>
<td>429</td>
<td>1.00E+02</td>
<td>429</td>
<td>1.00E+02</td>
</tr>
</tbody>
</table>
How is JGI doing with all that currently?

- About half of BLAST processes are not multithreaded

<table>
<thead>
<tr>
<th>Processes</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125509</td>
<td>1</td>
</tr>
<tr>
<td>1327906</td>
<td>8</td>
</tr>
<tr>
<td>227298</td>
<td>4</td>
</tr>
<tr>
<td>132955</td>
<td>2</td>
</tr>
<tr>
<td>84781</td>
<td>32</td>
</tr>
<tr>
<td>70815</td>
<td>16</td>
</tr>
<tr>
<td>7063</td>
<td>5</td>
</tr>
<tr>
<td>3151</td>
<td>10</td>
</tr>
</tbody>
</table>
Thoughts on BLAST databases

- We see some users with small, specific databases

Exponential increase in jobs, filesystem access, output files...
Database reads

• While extending and evaluating number of HSPs is the most severe bottleneck, accessing the database is a significant one.

• Disk access time can be significant, because current version does not read all of db into memory, even if enough memory available to do so.

• Large job arrays may cause GPFS slowdowns.

• Reduce disk access time by:
  – Never run from $HOME directory
  – Stage your database directly to the compute node /scratch
  – Use one of our newly pre-staged standard dbs
Pre-staged standard BLAST databases

• Mendel nodes have more /scratch space, and we are copying some frequently-used databases to
  – /scratch/blastdbs
• Working with users on plan to keep updated

• Current databases:
  - gcontam
  - mito.nt
  - refseq.archaea
  - refseq.fungi
  - refseq.plant
  - refseq.viral
  - unite
  - nt
  - refseq.bacteria
  - refseq.mitochondrion
  - refseq.plasmid
Summary of recommendations

• Ask yourself if BLAST is the best tool for the job, and consider which BLAST executable
• Use a current blast executable
• Use an appropriate E-value for your desired output
• Limit the number of output sequences
• Use multiple sequence inputs
• Use multithreading
• Combine your databases
• Use pre-staged databases, or stage your own