Explore Spark for Metagenome assembly

Zhong Wang, Ph.D.
Group Lead, Genome Analysis
Environmental microbial communities are complex

>90% of the species haven’t been seen before
Decode Metagenome

Genome \sim=\text{Book} \quad \text{Metagenome} \sim=\text{Library}

Library of Books $\rightarrow$ Shredded Library $\rightarrow$ “reconstructed” Library

Metagenome

Shotgun Sequencing \quad \text{Short Reads}

Billions of pieces Terabytes in size “Big Data”

Assembled Genomes
The Ideal Solution

- Easy to develop
- Robust
- Scale to big data
- Efficient
2009: Special Hardware

$1M
Only scale up to ~100Gb

Jeremy Brand @JGI
FPGA @Convey
2010: MP/MPI on supercomputers

- Experienced software engineers
- Six months of development time
- One task fails, all tasks fail

Problems:

Fast, scalable

MPI version
412 Gb, 4.5B reads
2.7 hours on 128x24 cores
NESRC Supercomputer

Rob Egan @JGI
2011: Hadoop/Map Reduce framework

- **Google MapReduce**
  - Data Parallel programming model to process petabyte data
  - Generally has a map and a reduce step

- **Apache Hadoop**
  - Distributed file system (HDFS) and job handling for scalability and robustness
  - Data locality to bring compute to data, avoiding network transfer bottleneck
Programmability: Java vs Pig

finding out top 5 websites young people visit

Users = load 'users' as (name, age);
Fltrd = filter Users by age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
COUNT(Jnd) as clicks;
Srtgd = order Smmd by clicks desc;
Top5 = limit Srtgd 5;
store Top5 into 'top5sites';
2013: BioPig

BioPig:
- Blaster
- Assembler
- Extender

BioPig: 61 lines of code
MPI-extender: ~12,000 lines (vs 31 in BioPig)

Scalability

Karan Bhatia, Henrik Nordberg, Kai Wang
Challenges in application

• 2-3 orders of magnitude slower than MPI
• IO optimization, e.g., reduce data copying
• Some problems do not easily fit into map/reduce framework, e.g., graph-based algorithms
• Runs on AWS, but cost $$$ if not optimized
Optimizing BioPig

Time saving (60GB)

Baseline: 100 minutes
Compression: 8 minutes
Block Size: 17 minutes
Map Side Spill: 3 minutes
JVM GC: 11 minutes
SlowStart: 8 minutes
Final: 53 minutes

Still very low efficiency!

3X Speed up

Baseline
Tuned
EMR

Lizhen Shi, Weikuan Yu @FSU
Addressing big data: Apache Spark

- New **scalable** programming paradigm
  - Compatible with Hadoop-supported storage systems
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell
Goal: Metagenome read clustering

• **Data characteristics:**
  – Total data size typically 100Gb – 1Tb
  – >1 billion short pieces (reads, each 100-200bp)
  – >1,000 different species, some species are more similar than others
  – Sequence errors 1-2%

• **Proposed approach: Divide-and-conquer**
  – Cluster reads from each genome (Clustering)
  – Assemble each cluster in parallel (Assembly)
Read clustering with Spark: idea

- **Local information: overlap**

  Spark *Metagenome*

  *Metagenome Assembler*

- **Global information: covariance**

<table>
<thead>
<tr>
<th>Sample1</th>
<th>Sample2</th>
<th>Sample3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Spark</td>
<td>2 Spark</td>
<td>10 Spark</td>
</tr>
<tr>
<td>1 Metagenome</td>
<td>2 Metagenome</td>
<td>10 Metagenome</td>
</tr>
<tr>
<td>1 Assembler</td>
<td>2 Assembler</td>
<td>10 Assembler</td>
</tr>
<tr>
<td>10 de</td>
<td>2 de</td>
<td>5 de</td>
</tr>
<tr>
<td>10 bruijn</td>
<td>2 bruijn</td>
<td>5 bruijn</td>
</tr>
</tbody>
</table>
Read clustering with Spark: preprocess

Sample 1

- Reads RDD: {Read_ID => Sequence}
- k-mer emission
- k-mer RDD: {kmer => [list(Read_IDS)]}
- Filter min <= count <= max
- Filtered k-mer RDD: {k-mer => [list(Read_IDs)]}

Spark RDD: Resilient Distributed Dataset

Data Filtering

Sample n

- Reads RDD: {Read_ID => Sequence}
- k-mer emission
- k-mer RDD: {kmer => [list(Read_IDS)]}
- Filter min <= count <= max
- Filtered k-mer RDD: {k-mer => [list(Read_IDs)]}

Union
Read clustering with Spark: core

Local Similarity

k-mer RDD: 
{k-mer => 
[\text{list(Read\_IDs)}]_1, 
......
\text{list(Read\_IDs)}]_n} 

Local similarity

ReadPairRDD: 
{\text{ReadPair => 
#shared-kmers}}

Global Similarity

count-normalization, k-SVD

Filter by cluster size

MLlib: SVD

GraphX

Local Edge

ReadGraph: 
{\text{Vertices: Read; 
Edges [local, global]}}

Global Edge

Graph Partition

Graph Partition #1 
Assembly\_by\_Overlap

... ...

Graph Partition #m 
Assembly\_by\_Overlap

Read Graph
Species:
- 6 bacterial species
- Synthetic communities with random proportions of each

Data: single genome sequence data (synthetic & real reads)
Cluster evaluation criteria: NMI

NMI: normalized mutual information

**Mutual Information**: How pure the different clusters are

\[
NMI(\Omega, \mathcal{C}) = \frac{I(\Omega; \mathcal{C})}{[H(\Omega) + H(\mathcal{C})]/2}
\]

**Entropy**: Penalizes having small clusters

Testing Environments

- **Local**
  - *Algorithm development*
  - 32-core
  - 256GB memory

- **HPC-Lawrencium**
  - *Small scale analysis*
  - CPU: INTEL XEON E5-2670
  - 16-core per node
  - 64GB memory per node
  - Infiniband FDR

- **NERSC-Cori**
  - *Large scale analysis*
  - CPU: Cray Haswell
  - 32-core per node
  - 128GB memory per node
  - Cray Aries high-speed interconnect with Dragonfly topology
Local Similarity

In ideal situation (no errors, no repetitive sequences, sufficient sequence coverage): read clustering with local similarity works perfectly.

With real-world situations where:

Sequencing coverage is low, many small clusters may form from a same genome, leads to False Negatives

Different genome share sequences, they can fall into the same cluster, leads to False Positives.
Some performance metrics

Spark Runtime on 16-node on Lawrencium

Needs 500-700X of memory – optimization is needed

Xiandong Meng @JGI
# Global Similarity: input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>No samples</td>
<td>1-1000</td>
</tr>
<tr>
<td>K-mer Length</td>
<td>20-50</td>
</tr>
<tr>
<td>K-Means Clusters</td>
<td>10-500</td>
</tr>
<tr>
<td>Eigen K-mers to sample</td>
<td>1-10,000</td>
</tr>
<tr>
<td>Eigen Reads to sample</td>
<td>100-60,000</td>
</tr>
<tr>
<td>Global Weight</td>
<td>0-150</td>
</tr>
<tr>
<td>Power Iteration Clusters</td>
<td>10-150</td>
</tr>
<tr>
<td>Power Iteration Steps</td>
<td>0-50</td>
</tr>
</tbody>
</table>
Exploring the parameter space

Predict Output Score from Input
Overall impression of Spark

- Easy to develop
- Robust
- Scale to big data
- Efficient
  - VS Hadoop/PIG
  - VS MPI
Acknowledgements

BioPig Team
Henrik Nordberg
Kai Wang
Karan Bhat @ Amazon
Lizhen Shi, Weikuan Yu @ FSU

Spark Team
Xiandong Meng
Jordan Hoffman

Lisa Gerhardt, Evan Racah
Shane Cannon @ NERSC

Gary Jung, Greg Kurtzer
Bernard Li, Yong Qin @ HPC