“Genome Browser”

- Service
  - Software
  - Data
  - Servers
  - People
    - Software Development
    - Data management
    - Support
    - Outreach/Training
“Genome Browser”

- Service
  - Software
    - JBrowse
  - Data
  - Servers
  - People
    - Data management
    - Support
    - Outreach/Training
Someone Else Does the Data Management

I have seen kent/src/hg/makeDb/doc/*
This talk

- Story of JBrowse
- Overview
- Demo
- Implementation
Traditional Web-based Genome Browsers
Traditional Web-based Genome Browsers
Tiling?
Tiling?
First prototype
Rendering Tiles - expensive

- D. melanogaster genome: ~120 million bases
- max zoom: 10 pixels/base
  - => 1.2 billion pixels wide, (US in google maps is only 35 million pixels wide)
  - 10s to 100s of pixels high,
- For the entire Drosophila genome, 10 tracks, all zoom levels
  - Space for tiles: 15gb
  - Rendering time: 15hours
Rendering tiles - optimizations

- Space
  - Hard-linking identical tiles

- Time
  - GD:
    - Memset for filling rectangles
    - Memcpy for creating tiles from larger image

- Space & Time: Render on demand
  - Still requires substantial up-front work

Our server-rendered approach was inherently expensive.
Client-side rendering?

- **SVG**
  - Not in IE
  - Scalable?
Client-side rendering?

- SVG
  - Not in IE
  - Scalable?
Client-side rendering: HTML

- Rectangles: work for genomic features
- HTML rectangles have fairly rich functionality
- Zooming: position/size in percentage units
Most genome browsers
JBrowse

- Almost no waiting for the server
- Cheaper/easier to run your own genome browser
Client-side advantages
(relative to our earlier prototype)

- Much less storage/CPU usage on the server
- Useful amounts of data can be cached on the client
  - Maybe even enable off-line usage
- Client can do a lot more
  - Highlight features
  - Show subsets of features
  - Edit features? e.g., gene model curation
- Can combine image-based, pre-rendered tracks side-by-side with client-rendered tracks
JBrowse

- Works (and tested) in IE 6+, Firefox 2+, Safari 3+, Chrome

- Data sources:
  - Feature flatfiles: GFF2/3, BED
  - Quantitative data: WIG
  - Next-gen: BAM
  - BioPerl Bio::DB databases
    - Bio::DB::GFF, Bio::DB::SeqFeature::Store, chado, DAS/1
This talk

- Story of JBrowse
- Overview
- Demo
- Implementation
Implementation

- Interbase!
- Caching
  - Useful amounts of data can be cached on the client
- (Relatively) simple installation
- Low CPU usage at request-time
  - HTTP server only serves static files, no CGI
Storage vs. Computation

Do work at request (read) time

Computation

Storage

Do work at write time

Computation

Storage

Assumption: read-heavy workload
Browser (coordinates everything, performs name lookups)
GenomeView (handles scrolling/zooming)

SequenceTrack
FeatureTrack
ImageTrack

(88% of the code in JBrowse)
Nested Containment Lists
Sorted by both start and end
Range query
Range query
Range query
Real feature data isn't like that
Sorted on start, but not on end
Containment
Nested Containment Lists

Nice Things About NCLLists

- Simple to implement
  - Just a recursive binary search + iteration
- Tree structured
  - Just like JSON (!)
- Fast
Scalability

Large numbers of features?
How to break up the data?
JBrowse uses NCLists
Lazy NCLists?
“fake” features
Lazy Loading in JBrowse
Lazy Loading in JBrowse

BAM example

- On one test data set:
  - 4.4 million features
  - 8 minutes to process
    - From 242 megabyte BAM file
    - Not paired-end
  - Used 400 megabytes of RAM
  - 330 megabytes on disk (without sequence)
  - Compresses down to 80 megabytes
Other approaches to lazy loading

- Heng Li (SAMTools)
  - Binning, linear index
- Jim Kent (BigBed/BigWig)
  - R-Trees
- JBrowse javascript client can't use them directly
  - Can convert them to JSON
  - Or, potentially, access them through a proxy
Sequence Metadata (JSON)

Bio::DasI
- Bio::DB::GFF
- Bio::DB::SeqFeature::Store
- Bio::DB::Das::Chado

GFF, BED, FASTA

Per-track Nested Containment List (JSON)

Sequence Chunks (text)

Track Metadata (JSON)

Lazily-loaded PATRICIA trie with all names (JSON)

PNG image tiles (multiple zoom levels)

Wiggle file

biodb-to-json.pl

flatfile-to-json.pl

prepare-refseqs.pl

generate-names.pl

CGi

Config File (JSON)

Server
Wiggle tracks: pre-rendered

- Only rendered up to 1 base per pixel
- Implemented in C++
- ~12 min to generate tiles for Dmel conservation track (1 data point per base)
  - => ~1min per 10 million bases
- Wiggle tiles compress well
  - ~5 bytes/base, half of which is filesystem overhead
- They could also be rendered on the fly
Bio::DasI
- Bio::DB::GFF
- Bio::DB::SeqFeature::Store
- Bio::DB::Das::Chado

Per-track Nested Containment List (JSON)
+ Track Metadata (JSON)

Lazily-loaded PATRICIA trie with all names (JSON)

Sequence Metadata (JSON) + Sequence Chunks (text)

PNG image tiles (multiple zoom levels)

Wiggle file

Wiggle file

Server

Config File (JSON)

bio db-to-json.pl

flatfile-to-json.pl

prepare-refseqs.pl

generate-names.pl
Name/ID searching: Trie

- Shares prefixes among a set of strings
Name/ID searching: Trie

- Subtries are lazily loaded
Summary

• Compared to existing web-based genome browsers, JBrowse:
  – Moves work from server to client
  – Moves work from read-time to write-time

• Caching
  – Offline usage?

• Scalability: it works, still some bugs

• Intended to fit in as a component of a larger system
Thanks

- Ian Holmes
- Andrew Uzilov
- Lincoln Stein
- Chris Mungall
- GMOD
  - Scott Cain
  - Dave Clements
- BioPerl
- NHGRI
- Users
  - Brenton Graveley

jbrowse.org
Pre-rendering isn't totally crazy

000010000    000010000
010011010    010001010
111111011    101100001
111111111    000000100
111111111    000000000