Database Engineering Document

Assembly database architectures.

1. Messages.
    a. Designed by the early Assembly Team.
    b. A message file is a collection of structured records. Files usually contain multiple record types, including an audit record at the beginning and end.
    c. A message file may be binary or text. Text messages are for prototyping, while binary messages are for production. Programs assume messages are binary by default, and optionally read text messages. In practice, most messages are text because it is useful to be able to read the data during post-mortems. The assembly pipeline is now called 'Proto I/O' even when it is not in prototype.
    d. Assembly pipeline input files.
        i. Message files.
        ii. *.frg contains fragments and links.
        iii. The input is generated by the Pre-Assembly Team, who reads the data from an Oracle database called PRD_WQ.
    e. Assembly pipeline output files.
        i. Message files.
        ii. Produced by the Terminator program.
        iii. *.asm contains the entire assembly graph.
        iv. The output is passed to the Chrom Team, who writes the data into an Oracle database called PRD_HUM.
        v. Note the inefficiency. Terminator reads internal files, generates a text file, which is transformed into a bulk load file, and loaded into Oracle. Terminator probably could load Oracle directly.
2. Gatekeeper store.
    a. Holds fragment metadata (no large fields) and link data.
    b. Uses the AS_PER_genericStore architecture. Developed by Saul Kravitz.
    c. The fragment record structure contains a pointer to a link, and unfortunately the Gatekeeper program must update this pointer after writing the fragment; this extra I/O is very noticeable. **Engineering project:** redesign the fragment file so that records get written once. Move the fragment-to-link information to a separate cross table that also gets written once. Make sure the downstream effect of reading the extra cross table is not onerous.
    d. Written to by the Gatekeeper program.
    e. Read by Screener, Terminator, CGW.
3. Fragment store.
    a. Holds fragment large fields (sequence, quality, source) and some metadata in 3 files.

        i. For each fragment, the index contains some fragment metadata. This is duplicated information; all fragment metadata is retained in the Gatekeeper store. The index also contains
            1. Fagment internal ID (IID), a 32-bit integer used only by the genome assembler.
            2. Offset into the sequence file, a partitioned 64-bit integer whose first byte is the file number.
            3. Offset into the source file, a partitioned 64-bit integer whose first byte is the file number.
        ii. For each fragment, the sequence file contains sequence and quality data, both strings. Sequence is the string of called bases determined by the sequencing machine. Quality is the score assigned to each called base by the software associated with the sequencing machine.
        iii. For each fragment, the source file contains a free text blob. Meant to contain only human readable notes, it now also contains essential information parsed by the assembler.
    b. Uses the AS_PER_genericStore architecture. Developed by Saul Kravitz.
    c. The fragment order is the same as the order in the input file from Pre-Assembly; this is a sore point for Unitigger.
    d. Written to by the Populator.
    e. Read by Overlapper, Screener, CGB (?), Consensus, CGW, Terminator.
    f. This store was meant to replace all fragment message files. In practice, some fragment messages persist. For instance, OFG and AFG.
4. Overlap store.
    a. Holds known overlaps between fragments.
    b. Has its own architecture, built on the VA (variable array) module. Developed by Art Delcher.
    c. Written to by the Overlapper program.
    d. Read by the Reaper and FGB components of the Unitigger program.
5. Unitigger store.
    a. Written and read by components of Unitigger: Repair breakers, Bubble smoothing, Reaper, FGB, CGB.
    b. Uses its own architecture, built on the VA (variable array) module. Developed by Clark Mobarry.
    c. The genericStore was deemed unacceptable as an architecture for these reasons
        i. Unitigger uses 64-bit addressing. Generic store could not handle fwrite()s of steams whose length exceeded 2^{30} bytes. **[No longer a problem?]**
        ii. Generic store could not resort the data on a virtual field. Unitigger assigns its own internal IDs to fragments and, for speed, must read them back sorted by this ID.

   d. Represents a graph where vertices are fragment ends, and edges are overlaps. The edge between ends of the same fragment is implicit; both ends have the same fragment ID.
   e. Records are stored in the order they are most likely to be used. This is a feature saves time during de-chording. Programs can read batches of records into memory and process them with mostly local memory access.
6. Partitioned Fragment Store.
   a. Written and read by the Consensus program. The '**consensus –S**' option takes a partition number. The '**consensus –B**' option sets the approximate number of fragments per partition.
   b. Developed by Karin Remington, Saul Kravitz, Clark Mobarry.
   c. Used on large genomes only. Allows Consensus to run many times in parallel, with one data partition per processor.
   d. The API and implementation are borrowed from, and intentionally similar to, the Fragment Store. Functions in the Partitioned Fragment Store API have one additional parameter for partition number.
   e. The number of fragments per partition is input as a command-line argument in '**consensus –B**'. The constraint is the number of fragment sequences that Consensus can load into memory. Consensus usually runs on computers with 2GB RAM (on the compute farm managed by LSF software), and read lengths average 500 bp in length, so it is not surprising that the number 500,000 has worked in practice. However, since read lengths will increase as technology improves, this number should be calculated (automatically?) before each run. [**Potential engineering project]**
   f. The partitioned store is created by an **O**(n) bin sort. While the partitions (bins) are ordered and mutually exclusive, the fragments within them are unsorted.
   g. Consensus reads into memory all the sequence data for all the fragments in a bin. It might be possible to lower the memory requirement on Consensus **[Potential engineering project]**. If fragments within partitions were sorted, Consensus could load the sequences for local fragments only, in a sliding window. On the other hand, the time cost of extra sort() and fseek(), vs. the current memory map, might outweigh the space savings.

Here is the flow of fragments. **[Get this right.]** FRG -> IFG -> SFG -> OFG -> AFG -> ASM. Gatekeeper reads FRG files. Screener and populator use SFG files. Unitigger uses OFG files, which is like IFG minus the sequence and quality. CGW uses AFG files, which again contains the UID.

The final output to text is a terrible waste. Our binary stores are combined with our text messages to produce a text ASM file. Chrom Team loads Oracle from the ASM file. A better design would load assembler's output directly into the database. Utilities for checking the ASM output would have to be changed to read the database or load files.

Saul Kravitz developed one persistent storage architecture. The stores are well tested and bug free. The stores are robust to scaling and porting. The stores have been tailored to the assembly program, and are very fast, much faster than Oracle could ever be. The stores are purposely occluded to force access via the API called AS_PER_genericStore. The stores do not support concurrent write access. The AS_PER directory contains some utility programs for working with stores. Saul claims the stores were developed in one week.

AS_PER utility programs for generic stores.

1. Dump.
2. **[List these!]**

Engineering project: Conditional fseek() in generic store.

The generic store performs a precautionary fseek() before every fwrite() inside an append function. This is in case the file pointer had been moved (by an fread()) between appends.

This has caused excessive buffer flushing and lseek() system calls that overflow the NFS cache when run on an NFS-mounted disk.

A more efficient design would only fseek() when necessary. To implement this change, add a file position field to the struct StoreStat in AS_PER_genericStore.h. Then in AS_PER_genericStore.c, modify the append(), get(), and set() methods to update this field after every fread(), fwrite() or fseek(). Then modify the append() methods to check the value before doing an fseek(). Saul Kravitz says it is safe to assume the genericStore module is the exclusive conduit to store files, that is, that no programs access the files directly.

Oracle

Less than a year ago, IR considered using Oracle for assembly. Saul Kravitz and Mike Flanigan were the investigators. The test case simulated the Consensus section of CGW. Oracle assigned people to the experiment. Oracle was not able to deliver a sufficiently fast database. At the same time, IR developed its own alternative and lost interest in Oracle. IR's solution was to partition the fragment data into smaller fragment stores. Each partition was self-contained, allowing each parallel CGW to load all its data into memory.