technische universität
dortmund

Bachelorarbeit

Erweiterung von CoGaDB um ein Werkzeug zur Visualisierung von Genomdaten

John Sarrazin
Oktober 2015

Gutachter:
Prof. Dr. Jens Teubner
Prof. Dr. Gunter Saake

Technische Universität Dortmund
Fakultät für Informatik
Datenbanken und Informationssysteme (LS 6)
http://dbis.cs.tu-dortmund.de/cms/en/home/

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Die Genomanalyse ist ein wichtiger Bestandteil aktueller Forschung. So können Risiken für Krankheiten bei Menschen bestimmt oder gewollte Eigenschaften von Pflanzen in der Agrarwirtschaft untersucht werden. Durch die Entwicklung von *Next-Generation-Sequencern* Anfang des 21. Jahrhunderts wurde es möglich, mit wenig Kosten und Zeit eine große Menge an Genomdaten zu sequenzieren. Weil ein einzelnes Genom oft mehrere Gigabyte groß ist, stellt die Datenverarbeitung von Genomdaten eine bedeutende Herausforderung der Informatik dar. In der derzeitigen Praxis werden diese Genomdaten in einfachen Dateien gespeichert. Werkzeuge und Algorithmen, die teilweise zu Pipelines hintereinander geschaltet werden können, übernehmen einzelne Aufgaben der Genomanalyse. Ein Nachteil dieses Ansatzes ist, dass auf diese Weise ein sehr hoher Overhead im Datenmanagement entsteht. Ein weiterer Nachteil ist die fehlende Interaktivität der Werkzeug-Pipelines. Ein Ansatz, der diese Nachteile umgehen soll und ein flexibles Datenmanagement von Genomdaten ermöglicht, ist der Einsatz von relationalen Datenbankmanagementsystemen (DBMS).

1.2 Probleme

Es existieren bereits einige Ansätze in der aktuellen Forschung, die den Einsatz von DBMS im Kontext der Genomforschung untersuchen. Meistens wird dabei jedoch nicht untersucht, wie eine Einführung dieses Ansatzes in der Praxis umzusetzen ist. Die Einführung einer neuen Technologie hat in der Praxis nur Erfolg, wenn eine Rückwärtskompatibilität zur vorher genutzten Technologie gegeben ist. Jeder Genomforscher bzw. jedes Institut hat seine eigenen Methoden und Werkzeuge in der Genomanalyse. Eine Umstellung auf eine neue Technologie muss sicherstellen, dass diese immer noch funktionieren. Aus diesem Grund ist es notwendig, dass bei Einführung eines Ansatzes zur Speicherung und Analyse von Genomdaten in relationalen Datenbanksystemen sichergestellt wird, dass eine Anbin-

dung an bisher in der Praxis eingesetzte Werkzeuge und Werkzeug-Pipelines möglich ist. Diese Anbindung sollte mit minimalem Aufwand implementiert werden können und keinen großen Performance- bzw. Ressourcen-Overhead erzeugen.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, den vielversprechendsten Ansatz zu entwickeln, der möglichst alle bestehenden Werkzeuge der Genomforschung in ein DBMS integriert. Dabei sollen folgende Forschungsfragen beantwortet werde:

1. *Wie lassen sich möglichst alle bestehenden Werkzeuge am vielversprechendsten an ein DBMS anbinden?*
2. *Wie lässt sich eine solche Anbindung implementieren?*
3. *Wie lässt sich eine solche Anbindung evaluieren?*
4. *Wie wird eine solche Anbindung im Vergleich zum aktuellen Stand der Technik bewertet?*
5. *Woraus resultiert diese Laufzeit?*

Um diese Forschungsfragen zu beantworten, werden mögliche Integrationsansätze entwickelt und anhand von definierten Bewertungskriterien miteinander verglichen. Der vielversprechendste Ansatz wird anschließend konzeptionell umgesetzt und an einem konkreten DBMS implementiert. Anhand einer Evaluation wird untersucht, ob der entwickelte Ansatz korrekt, effizient und skalierbar umgesetzt werden kann.

1.4 Struktur der Arbeit

Die Arbeit gliedert sich wie wie folgt:

Kapitel 2 In Kapitel 2.1 werden zunächst einige für das Verständnis der Arbeit wichtige Grundbegriffe und Prozesse der Genomforschung beschrieben. Weiter werden in Kapitel 2.2 das SAM-Format und SAMtools vorgestellt. Kapitel 2.3 beschreibt Grundlagen und Probleme des Ist-Zustandes vom Datenmanagement in der Genomforschung, d. h. dem Einsatz von einfachen Dateien und Werkzeugketten. Das Kapitel schließt mit der Beschreibung, wie Genomdaten in einem DBMS gespeichert werden können.

Kapitel 3 Das dritte Kapitel diskutiert verschiedene Ansätze, die eine Anbindung der Datenbanktechnologie an bestehende Werkzeuge bzw. Workflows der Genomforschung ermöglichen. Diese werden in Kapitel 3.1 zunächst vorgestellt. Anschließend wird in Kapitel

3.2 eine Bewertungsskala, sowie Bewertungskriterien entwickelt, mit denen die vorgestellten Ansätze in Kapitel 3.3 vergleichend evaluiert werden. Das Kapitel schließt damit, dass der vielversprechendste Ansatz ausgewählt wird.

Kapitel 4 Der vielversprechendste Integrationsansatz wird im vierten Kapitel zunächst konzipiert (Kapitel 4.1) und anschließend am Beispiel von CoGaDB und IGV exemplarisch umgesetzt (Kapitel 4.2). Zum Schluss folgt eine Zusammenfassung und ein Ausblick im Bezug auf die Umsetzung des Integrationsansatzes in Kapitel 4.3.

Kapitel 5 Im fünften Kapitel wird die Umsetzung des Integrationsansatzes anhand der Bewertungskriterien Korrektheit (Kapitel 5.1), Laufzeit (Kapitel 5.2) und Skalierbarkeit (Kapitel 5.3) evaluiert.

Kapitel 6 Die Arbeit endet mit einer Zusammenfassung der wichtigsten Erkenntnisse, einem generellen Fazit und einem Ausblick.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen geschaffen, die für das weitere Verständnis dieser Arbeit von Bedeutung sind. Zunächst wird eine Einführung in die Genomforschung gegeben und wichtige Grundbegriffe erläutert. Anschließend wird das Sequence Alignment/Map Format (SAM-Format) und das Softwarepaket SAMtools vorgestellt. In Kapitel 2.3 wird der Ist-Zustand der praktischen Genomforschung erklärt. Diese funktioniert auf Basis von einfachen Dateien und hintereinandergeschalteten Werkzeugketten. Das Kapitel schließt mit der Beschreibung der Genomforschung auf Basis von relationalen Datenbankmanagementsysteme (DBMS).

2.1 Grundlagen der Genomforschung

Dieser Abschnitt erläutert die für das Verständnis dieser Arbeit relevanten Grundlagen und Grundbegriffe der Genomforschung.

2.1.1 Grundbegriffe

Genom Unter dem Begriff Genom wird das gesamte Genmaterial eines Lebewesens gefasst [27, S.6]. Es enthält also dessen sämtlichen Erbinformationen [24, S.26].

DNA Desoxyribonukleinsäure (DNA) ist ein in Molekül im Körper von allen Lebewesen. In ihm sind die Geninformationen des jeweiligen Lebewesen codiert [9]. Die DNA besitzt üblicherweise zwei Stränge, die in Form einer Doppelhelix angeordnet sind. Jeder dieser Stränge besteht aus einer Folge von vier Einzelbausteinen, die sich jeweils aus einem Zucker, einer Nukleinbase (im Folgenden Base genannt) und einem Phosphat [24, S.4] zusammensetzen.

Basen-Paare Jedes DNA-Element enthält eine von vier Basen - Adenin (A), Thymin (T), Guanin (G) und Cytosin (C). Aufgrund der Anordnung der beiden Stränge in Form

einer Doppelhelix bilden jeweils zwei Basen ein Paar. Entweder bilden Adenin und Thymin oder Guanin und Cytosin ein Basen-Paar [24, S.5]. Da eine Base in einem Paar die andere impliziert, kann jedes Paar durch die Angabe einer Base eindeutig kodiert werden [9].

DNA-Sequenz DNA-Sequenzen sind Folgen der vier Buchstaben A,C,G und T. Jede dieser Buchstaben repräsentiert eine der vier zuvor beschriebenen Basen [24, S.4]. Durch DNA-Sequenzen können DNA-Informationen und Genome digital gespeichert werden [13].

Referenzgenom Das Referenzgenom ist eine Repräsentation des Durchschnittsgenoms einer Spezies. Es wird von Wissenschaftlern anhand vieler Genomdaten entwickelt und kann als eine Art Landkarte dienen, um DNA-Sequenzen an der richtigen Stelle anzuordnen (*Sequence Alignment*). Außerdem können Abweichungen vom Referenzgenom, sogenannte Varianten, Indikatoren für Eigenschaften eines individuellen Lebewesens sein [13]. Abschnitt 2.1.2 gibt eine genauere Übersicht über die Themen *Sequence Alignment* und *Variant-Calling*.

Sample-Genom Das Sample-Genom ist das konkrete Genom eines Lebewesen, dass man untersuchen möchte. Dieses wird für die Analyse mit dem Referenzgenom verglichen, um Schlüsse auf das Lebewesen zu ziehen.

Contig Als Contig wird ein Teil des Referenzgenoms bezeichnet.

Read Reads sind kleine, überlappende Teilstücke des Sample-Genoms, die beim Sequenzieren entstehen [10]. Details zum Sequenzieren von Genomen befinden sich in Kapitel 2.1.2.

Alignment Ein *DNA-Alignment* bezeichnet die Anordnung der Reads im Verhältnis zum Referenzgenoms. Details dazu befinden sich in Kapitel 2.1.2.

Phred-Quality-Score Der *Phred-Quality-Score* ist das Standardformat, um die Alignment-Qualität in der Genomforschung zu bewerten [8]. Der Wert wird durch folgende Formel berechnet:

$$Q_{PHRED} = -10 \log_{10}(\varphi)$$

φ steht dabei für eine Fehlerwahrscheinlichkeit. Der *Phred-Quality-Score* wurde mit dem Ziel entwickelt, sehr kleine Werte ($\leq 0,01$) gut unterscheiden zu können, da diese für die Genomforschung die wichtigsten sind. Bei einer Fehlerwahrscheinlichkeit von $1/1000$, ergibt sich zum Beispiel ein Qualitätswert von 30 [2]. In der Genomforschung wird es beispielsweise dafür eingesetzt, um die Qualität von Basen oder Alignments zu bewerten.

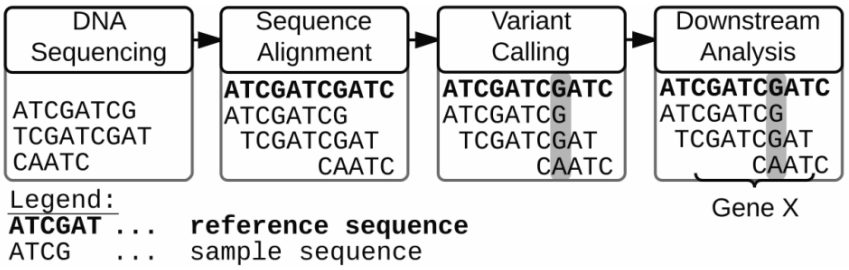

Abbildung 2.1: Minimale Genomanalyse-Pipeline [13]

2.1.2 Minimaler Ablauf einer Genomanalyse

Eine Genomanalyse besteht mindestens aus den vier Phasen *DNA-Sequencing*, *Sequence Alignment*, *Variant Calling* und *Downstream Analysis* [13]. Abbildung 2.1 (übernommen aus [13]) zeigt einen Überblick über die minimalen Schritte einer Genomanalyse.

1. Phase: DNA-Sequencing In der ersten Phase wird die DNA in DNA-Sequenzen umgewandelt, sodass sie digital gespeichert werden können. Aufgrund der technischen Beschränkungen von DNA-Sequenzierern kann das Genom nicht am Stück sequenziert werden. Stattdessen ist das Ergebnis der ersten Phase eine Menge von kleinen überlappenden DNA-Sequenzen, sogenannten Reads [25, 14].
Durch die Entwicklung von *Next-Generation-Sequencern* Anfang des 21. Jahrhunderts wurde es möglich, eine große Menge an Genomdaten mit wenig Kosten und Zeit zu sequenzieren. Die Hauptkosten der Genomforschung liegen dadurch nicht mehr bei der Sequenzierung, sondern vielmehr bei der Analyse der Genomdaten [23].

2. Phase: Sequence Alignment Die Reads aus der ersten Phase müssen im zweiten Schritt richtig zusammengesetzt werden, um die ursprüngliche DNA-Sequenz aus den Reads zu rekonstruieren [9, 14]. Dafür wird das Referenzgenom als eine Art Landkarte benutzt und die Reads im Verhältnis zum Referenzgenom angeordnet. Weil die Datenmengen sehr groß sind, braucht es besonders effiziente Algorithmen, um die Reads anzuordnen. Diese basieren entweder auf Hash-Tabellen oder suffix/prefix-Berechnungen [21].

3. Phase: Variant Calling Die dritte Phase besteht darin, Unterschiede zwischen dem Referenzgenom und dem Sample-Genom mit Hilfe von *Variant Calling* zu finden [29]. Dabei werden die Sequenzen des Sample-Genoms Schritt für Schritt mit denen des Referenzgenoms verglichen. Weil das Sample-Genom aus vielen überlappenden Reads besteht, sind an eine Base des Referenzgenoms meist mehrere Basen des Sample-Genoms angeordnet [12]. Daher muss ausgewählt werden, welche der Basen für den Vergleich verwendet wird. Mögliche Auswahlkriterien sind die Basenqualität [22] (vgl. Abschnitt 2.1) oder die relati-

ve Häufigkeit eines Basentyps [29]. Pabinger et al. geben einen Überblick über vorhandene Werkzeuge für *Variant Calling* [31].

4. Phase: Downstream Analysis Die Stellen, an denen Varianten gefunden wurden, werden in der vierten Phase anhand einer *Downstream Analyse* näher untersucht. Varianten können unterschiedliche Auswirkungen haben. Eine Variante kann keine Auswirkungen haben, aber auch Eigenschaften signifikant verändern [4]. Ein Beispiel für eine signifikante Änderungen wird von Komatsuda et al. beschrieben. Eine Mutation einer Gerstensorte kann dabei die Anzahl der Stacheln verändern [19]. Weil diese Phase von der Expertise und Interpretation von Genomforschern abhängig ist, werden oft Visualisierungswerkzeuge wie der Integrative Genomics Viewer[1] (IGV) eingesetzt [13]. Unter Downstream-Analyse werden alle weitergehenden Analysen gefasst. Oft werden dafür noch externe Zusatzinformationen benötigt [9].

2.2 SAM-Format und SAMtools

Das SAM-Format wurde mit dem Ziel entwickelt, ein universelles Format für angeordnete Genomdaten zu sein und somit als Schnittstelle zwischen *Sequence Alignment* und *Downstream Analysis*. Um diese Ziele zu erreichen, muss das Format in der Lage sein, das Ergebnis aller Alignment-Werkzeuge zu speichern und zu allen Sequenztypen kompatibel sein [20]. Eine Datei im SAM-Format ist eine tabulatorgetrennte Textdatei, welche sich in einen Header- und einen Alignment-Bereich gliedert. Der Header ist zwar optional, aber wenn einer angegeben wird, muss dieser vor dem Alignment-Bereich sein [39]. Das SAM-Format skaliert für Datenmengen mit mehr als 10^{11} Basenpaare und ist trotzdem flexible genug, um die Informationen von vielen verschiedenen DNA-Sequenzierer und Alignment-Werkzeuge zu speichern [20]. Weil dieses Format eine sehr wichtige Rolle für diese Arbeit spielt, wird es im Folgenden detailliert erläutert. In den folgenden Unterkapiteln sind alle Informationen, falls nicht anders angegeben, der SAM-Format-Spezifikation [39] entnommen.

2.2.1 Header-Bereich

Der Header Bereich ist ein optionaler Zusatz des SAM-Formats. Hier können allgemeine Meta-Informationen über die SAM-Datei hinterlegt werden. Auch Informationen zu den Contigs gehören in diesen Bereich. Eine Header-Zeile beginnt mit einem @, gefolgt von einer speziellen Kennung, welche den Zeilentyp bestimmt [9]. Jeder Zeilentyp hat eine Menge von zugehörigen Feldern, durch die nach der *Kennung:Wert - Syntax* Informationen hinterlegt werden können. Einige dieser Felder müssen für jede vorhandene Zeile des jeweiligen

[1] https://www.broadinstitute.org/igv/

Zeilentyps angegeben werden. Dies sind die Pflichtfelder dieses Zeilentyps. Im Folgenden wird ein Überblick über die verschiedenen Zeilentypen des Header-Bereichs gegeben und dabei die wichtigsten Felder erläutert. In der Überschrift wird die jeweilige Kennung in Klammern angegeben.

Headerzeile (HD) Diese Zeile muss, falls sie angegeben wird, die erste Zeile in der SAM-Datei sein und darf außerdem höchstens einmal in der Datei vorkommen. In dem Pflichtfeld der Headerzeile muss mit der Kennung *VN* die Versionsnummer der Format-Spezifikation angegeben werden. Weiter kann durch die Kennung *SO* angegeben werden, wie die angeordneten Reads des Alignment-Bereichs sortiert sind. Wenn unter dieser Kennung zum Beispiel das Schlüsselwort *coordinate* angegeben wird, bedeutet dies, dass die Reads nach dem Namen des Contigs sortiert werden, an dem sie angeordnet sind. Die Reihenfolge wird dabei durch das Verzeichnis der Referenzsequenzen impliziert.

Referenzsequenz - Verzeichnis (SQ) In diesem Verzeichnis befinden sich Metainformation über die Contigs. Für jedes Contig des Referenzgenoms kann eine SQ-Zeile angelegt werden. Wenn in der Headerzeile die Sortierung mit *coordinate* angegeben wird, wird diese durch die Reihenfolge der SQ-Zeilen bestimmt. Die Pflichtfelder enthalten den Namen des Contigs unter der Kennung *SN* und die Länge des Contigs unter der Kennung *LN*. Die Länge eines Contigs wird definiert als die Anzahl der Basen im Referenzgenom, die zu dem Contig gehören. Weitere Informationen, wie eine MD5-Prüfsumme, URL oder Spezies kann in den optionalen Feldern hinterlegt werden.

Read-Gruppierung (RG) Manchmal ist es gewollt, dass Genomdaten von verschiedenen Quellen in der selben SAM-Datei sind. Weil es im Nachhinein möglich sein soll, die Datensätze zu unterscheiden, können Read-Gruppen angelegt werden. Über optionale Felder des Alignment-Bereichs kann jeder Read einer Read-Gruppe zugeordnet werden. Für jede Read-Gruppe muss eine RG-Zeile angelegt werden. Jede dieser Zeile hat zumindest einen eindeutigen Identifikator, der über das Pflichtfeld mit der Kennung *ID* angegeben wird. Über die optionalen Felder können noch weitere Informationen hinterlegt werden. Darunter zum Beispiel wann, wo und mit welchem Programm die Reads dieser Gruppe erzeugt wurden.

Programm-Zeilen (PG) Hier können Informationen zu den Programmen und Werkzeugen hinterlegt werden, die zur Erzeugung oder Aufbereitung der SAM-Datei eingesetzt wurden [9]. Für jedes Werkzeug wird eine PG-Zeile angegeben. Als Pflichtfeld bekommt jedes Werkzeug einen eindeutigen Identifikator, der durch die Kennung *ID* gesetzt werden kann. Optionale Felder ermöglichen es beispielsweise noch, Kommandozeilenparameter oder Versionsnummern der Werkzeuge anzugeben. Auch kann in dem Feld mit der Ken-

nung *PP* eine Referenz auf eine andere PG-Zeile gesetzt werden. Auf diese Weise kann man die Reihenfolge definieren, in der die Werkzeuge eingesetzt werden.

Kommentarzeilen (CO) In jeder CO-Zeile kann ein einzeiliger Kommentar hinterlegt werden. Dies sind die einzigen Zeilen, in denen die *Kennung:Wert - Syntax* nicht verwendet wird.

2.2.2 Alignment-Bereich

In dem Alignment-Bereich existiert für jeden Read genau eine Zeile. Dieser Zeile kann entnommen werden, wie der Read im Verhältnis zum Referenzgenom angeordnet ist [9]. Eine Alignmentzeile besteht aus elf Pflichtfelder, die in jeder Zeile vorhanden sein müssen und kann durch zahlreiche optionale Felder erweitert werden. Für manche Pflichtfelder existieren Default-Werte, die gesetzt werden können, falls die Information nicht vorhanden ist. Alle Felder sind tabulatorgetrennt.

QNAME-Feld In diesem Feld wird der Name des Reads hinterlegt.

FLAG-Feld In diesem Feld können weitere Informationen durch das Setzen von Bits hinterlegt werden. Jedes Bit steht für eine Eigenschaft. Wenn zum Beispiel das Bit 0x4 gesetzt ist, bedeutet dies, dass dieses Read nicht angeordnet ist.

RNAME-Feld Das RNAME-Feld enthält den Namen des Contigs, an den der Read angeordnet ist. Wenn das Referenzsequenz-Verzeichnis im Header-Bereich angegeben ist, muss es in diesem eine zu diesem Eintrag passende Zeile geben. Wenn im FLAG-Feld das 0x4-Bit gesetzt ist, also der Read nicht angeordnet ist, enthält dieses Feld den Default-Wert '*'.

POS-Feld Das Feld POS enthält den Positionswert der am weitesten links liegenden Basen, an die das Read angeordnet ist. Für Reads, die nicht angeordnet sind, enthält dieses Feld den Wert 0. Für das Beispiel in Listing 2.1[2] ist der POS-Wert also 5.

MAPQ-Feld Dieses Feld beschreibt den *Mapping-Quality* des Reads und wird durch einen *Phred-Quality-Score* angegeben (vgl. Abschnitt 2.1). Als φ wird in diesem Fall die Wahrscheinlichkeit dafür genommen, dass das Read an der falschen Stelle des Referenzgenoms angeordnet ist. Weil der Wert für das MAPQ-Feld vom Typ Integer ist, muss der *Phred-Quality-Score* noch auf den nächsten Integer gerundet werden. An der Formel ist zu erkennen, dass ein hoher QUAL-Wert bedeutet, dass die Fehlerwahrscheinlichkeit gering ist.

[2]Entnommen von http://genome.sph.umich.edu/wiki/SAM

Name	Symbol	Beschreibung
Match	M	Base passt an dieser Stelle
Insertion	I	Read hat an dieser Stelle eine zusätzliche Base
Deletion	D	Read fehlt an dieser Stelle eine Base des Referenzgenoms
Softclipping	S	Base nicht angeordnet, jedoch Teil von SEQ
Hardclipping	H	Base nicht angeordnet und nicht Teil von SEQ
Padding	P	Referenzgenom wurde an dieser Stelle aufgefüllt
Skipped Region	N	Bereich vom Referenzgenom wird übersprungen

Tabelle 2.1: Überblick über die CIGAR-Operationen

CIGAR-Feld Der CIGAR-String beschreibt die genaue Anordnung der einzelnen Basen eines Reads an das Referenzgenom. Dafür werden in der Standardvariante des CIGAR-Strings die drei Operationen *Match*, *Insertion* und *Deletion* definiert. Tabelle 2.1 gibt einen Überblick über die Operationen des CIGAR-Strings. Der CIGAR-String setzt sich nun aus Paaren von Nummern und Kennungen der oben genannten Operationen zusammen (z.B. 30M8I3M15D). Die Nummer steht für die Anzahl der hintereinander folgenden Basen, für die die nachfolgende Operation gilt.

```
RefPos:     1  2  3  4  5  6  7     8  9 10 11 12 13 14 15 16 17
Reference:  C  C  A  T  A  C  T     G  A  A  C  T  G  A  C  T  A
Read:                  A  C  T  A  G  A  A     T  G  G  C  T
```

Listing 2.1: Beispiel-Alignment

In dem Beispiel-Alignment in Listing 2.1 ist der zum Read gehörende CIGAR-String *3M1I3M1D5M*. Die ersten drei Basen A,C,T passen an die angegeben Stelle. Dann wird eine Base A eingefügt. Die nächsten drei Basen G, A, A passen wieder an die angegeben Stelle. Anschließend fehlt eine Base vom Referenzgenom, was durch eine Lücke im Read symbolisiert wird. Die letzten fünf Basen T, G, G, C, T passen an die angegebene Stelle. Wie an Position 14 zu sehen, bedeutet ein Match nicht zwangsläufig, dass die Base vom Read und vom Referenzgenom an dieser Stelle gleich sind.
Der erweiterte CIGAR-String fügt noch die vier Operationen *Softclipping*, *Hardclipping*,

Padding, *Skipped-Region* hinzu [20]. In Tabelle 2.1 befindet sich ein Überblick über diese Operationen.

RNEXT- und PNEXT-Feld Die Felder RNEXT und PNEXT sind gedacht, um zusammengehörende Reads zu verknüpfen. RNEXT enthält den RNAME-Wert und PNEXT den POS-Wert vom nächsten Read. Für RNEXT gibt es den Spezialwert '=', der gesetzt wird, wenn der nächste Read dem selben Contig angehört. Die Default-Werte von RNEXT und PNEXT sind '*' und 0. Diese werden gesetzt, wenn die Informationen nicht vorhanden sind.

TLEN-Feld Dieses Feld enthält die Anzahl der gemappten Basen eines Reads. Der Default-Wert ist 0 und wird gesetzt, wenn die Information nicht vorhanden ist.

SEQ-Feld Das SEQ-Feld enthält die Basen-Sequenz des Reads. Dafür werden die in Kapitel 2.1 beschriebenen Basen-Kennungen aneinandergereiht und als String gespeichert.

QUAL-Feld Im letzten Pflichtfeld wird die Qualität für jede Base des Reads gespeichert. Dazu wird zunächst der *Phred-Quality-Score* für jede Base berechnet (vgl. Abschnitt 2.1). Als φ wird die Wahrscheinlichkeit dafür genommen, dass die Base falsch ist. Die Ergebnisse werden dann mit Hilfe der ASCII-Tabelle in ein Zeichen umgewandelt und konkateniert. Damit die Zeichen alle in einem für Menschen sichtbaren Bereich sind, muss zuvor ein Offset von 33 addiert werden [8].

Optionale Felder Die optionalen Felder besitzen die Syntax *Kennung:Typ:Wert*. Die Kennung definiert, welche Informationen gespeichert werden. Der Typ legt fest, welchem regulären Ausdruck der Wert entsprechen muss. Im Wert wird die jeweilige Information gespeichert.

Beispiel Listing 2.2 zeigt ein Beispiel-Alignment und Listing 2.3 die dazugehöre SAM-Datei. Das Beispiel wurde aus der SAM-Format-Spezifikation [39] übernommen und für diese Arbeit gekürzt.

```
RefPos:      12345678901234  5678901234567890123456789012345
Reference:  AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT
Read1/1:          TTAGATAAAGGATA*CTG
Read2:          aaaAGATAA*GGATA
Read3:        gcctaAGCTAA
Read1/2:                                              CAGCGGCAT
```

Listing 2.2: Alignment für SAM-Beispiel

```
@HD VN:1.5 SO:coordinate
@SQ SN:Reference LN:45
Read1/1 99 Reference 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
Read2 0 Reference 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
Read3 0 Reference 9 30 5S6M * 0 0 GCCTAAGCTAA *
Read1/2 147 Reference 37 30 9M = 7 -39 CAGCGGCAT * NM:i:1
```

Listing 2.3: Zugehörige SAM-Datei

Das Beispiel-Alignment enthält drei Reads. Read1 wird dabei in die beiden Teile Read1/1 und Read1/2 aufgeteilt. Durch die Werte von RNEXT und PNEXT, wird in der SAM-Datei angegeben, dass die beiden Teile zusammengehören. Das '=' in RNEXT bedeutet, dass beide Teile dem selben Contig angehören. Read2 enthält ein Softclipping. Die ersten drei Basen sind in Listing 2.2 durch kleingeschriebene Buchstaben repräsentiert. Diese Basen sind zwar in dem Read angegeben, jedoch nicht angeordnet. Daher enthält der CIGAR-String in Listing 2.3 für diese Basen ein 'S' für Softclipping. Die QUAL-Werte sind nicht vorhanden, daher wird der Default-Wert '*' gesetzt. In Read1/2 wird in der SAM-Datei auch das optionales Feld *NM:i:1* angegeben. Dieses sagt aus, dass die Editierdistanz dieses Reads zum Referenzgenom gleich 1 ist.

2.2.3 BAM-Format

Das BAM-Format ist die binäre Umsetzung des SAM-Formats. Dieses Format wurde entwickelt um die Daten kompakter zu speichern und um eine bessere Performance zu gewährleisten. Eine BAM-Datei enthält dabei genau dieselben Informationen wie die entsprechende SAM-Datei [20].

2.2.4 SAMtools

SAMtools ist ein Werkzeug zum Durchsuchen und zur Manipulation von Dateien im SAM- oder BAM-Format [20]. SAMtools wird als C-Bibliothek und Kommandozeilenwerkzeug angeboten. SAMtools kann zum Beispiel SAM- oder BAM-Dateien sortieren und indizieren. Es können mehrere Dateien zusammengefasst werden und Dateien von einem Format in ein anderes umwandeln. Mit dem Befehl *view* können auch kleine Bereiche aus einer SAM-Datei herausgeschnitten werden. SAMtools ist unter der BSD[3]- und MIT-Lizenz[4] veröffentlicht. Eine vollständige Referenz über alle Funktionen von SAMtools findet sich der Internetseite von SAMtools[5].

[3] http://opensource.org/licenses/bsd-license.php
[4] http://opensource.org/licenses/mit-license.php
[5] http://www.htslib.org/doc/samtools.html

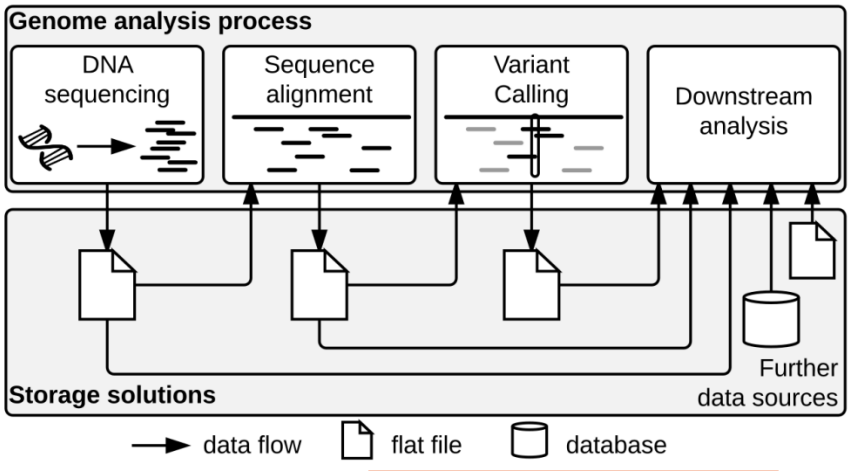

Abbildung 2.2: Minimale Genomanalyse-Pipeline mit Speicherlösungen [10]

2.3 Genomforschung auf der Basis von Dateien und Werkzeugketten

Dieser Abschnitt beschreibt, wie die Genomforschung auf Basis von einfachen Dateien und Werkzeugketten in der Praxis umgesetzt wird. Dabei werden zunächst die Grundlagen erklärt. Anschließend werden bestehende Probleme dieses Ansatzes erläutert.

2.3.1 Grundlagen

In der Praxis wird jeder Schritt der Genomforschung meist durch Kommandozeilenwerkzeuge umgesetzt. Diese nutzen einfache Dateien als Datengrundlage und werden mit Hilfe von Skripten zu Werkzeugketten hintereinander geschaltet, damit vollständige Analysezyklen durchgeführt werden können [13]. Die Skripte werden am Anfang des Analysezyklus durch Parameter konfiguriert und durchlaufen anschließend den gesamten Zyklus bis zum Ergebnis. Abbildung 2.2 von Dorok zeigt die Schritte einer minimalen Genomanalyse inklusive der eingesetzten Speichermechanismen [10]. Für jeden Schritt der Genomanalyse existieren viele verschiedene Werkzeuge und jeder Genomforscher hat ein eigenes Repertoire an Werkzeugen und Algorithmen, die er für individuelle Analysen hintereinander schaltet.

2.3.2 Probleme

Dieser Ansatz hat diverse Probleme, die sich überwiegend erst im Laufe der Zeit entwickelt haben.

Overhead im Datenmangement Durch den Einsatz von *Next-Generation-Sequencern* können in kürzester Zeit viele Genomdaten sequenziert werden. So kann jeder moderne Sequencer mehrere hundert Gigabyte Genomdaten pro Woche produzieren [7]. Durch ei-

ne solche Masse an Genomdaten entsteht ein erhöhter Overhead im Datenmanagement. Werkzeuge der Genomforschung müssen nicht nur Lösungen anbieten, die effizient sind, sie müssen gleichzeitig viele verschiedene Dateiformate unterstützen, damit sie mit möglichst vielen anderen Werkzeugen zusammenarbeiten können [10, 13].

Fehlende Flexibiltät Damit Werkzeuge hintereinander geschaltet werden können, muss sicher gestellt werden, dass diese kompatibel zueinander sind. Das wird meist dadurch erreicht, dass jedes Werkzeug in einer Kette das Output-Format des vorigen Werkzeugs unterstützt. Die meisten Werkzeuge sind jedoch ursprünglich nur dafür geschrieben worden, eigenständig zu arbeiten und nicht, um mit anderen Werkzeugen zu kommunizieren [4]. Daher können einzelne Werkzeuge einer bestehenden Werkzeugkette nicht ohne weitreichende Kenntnisse ausgetauscht werden. Auch das Anpassen der Skripts ist sehr fehleranfällig, weil die Werkzeuge oft speziell konfiguriert werden, damit die Werkzeugkette konsistent bleibt [13]. In der Praxis läuft es darauf hinaus, dass ohne große Kenntnisse nur auf bestehende Werkzeugketten zurückgegriffen werden kann.

Fehlende Interaktivität Ein Skript für einen Analysezyklus wird am Anfang einmal konfiguriert und gestartet. Anschließend läuft das Skript bis zum Ende durch. Es ist dabei nicht möglich, in den Analyseprozess individuell einzugreifen, also beispielsweise weiterführende Analysen für interessante Zwischenergebnisse durchzuführen [13].

Fehlende Reliabilität Für jedes Szenario existieren viele verschiedene Genomanalyse-Pipelines, die unterschiedliche Werkzeuge hintereinander schalten [4, 5, 32]. Allerdings existiert eine gewisse Abhängigkeit der Ergebnisse von den eingesetzten Werkzeugketten oder der Konfiguration. O'Rawe et al. zeigen beispielsweise, dass die Ergebnisse von *Variant Calling* auf demselben Datensatz sich für unterschiedliche Werkzeuge signifikant unterscheiden [30].

2.4 Datenmanagement von Genomdaten in relationalen DBMS

Ein vielversprechender Ansatz, um die in Kapitel 2.3.2 genannten Probleme zu lösen, besteht darin, die Analyseschritte der Genomforschung in einem DBMS durchzuführen [11, 13, 10]. In der aktuellen Forschung existieren viele Ansätze, einzelne Schritte der Genomforschung in einem DBMS durchzuführen oder diese durch Einsatz eines DBMS zu unterstützen [37, 35, 33]. Dorok et al. stellen außerdem einen Ansatz vor, sämtliche Schritte der Genomforschung in ein relationales Hauptspeicher-DBMS zu integrieren [11].
Um Genomdaten in einem relationalen DBMS speichern zu können, wird ein wohldefiniertes Datenbankschema benötigt. Das Genomdatenmodell nach Dorok et al. speichert die Genomdaten basenweise. Dabei werden nur Basen gespeichert, die an das Referenzgenom

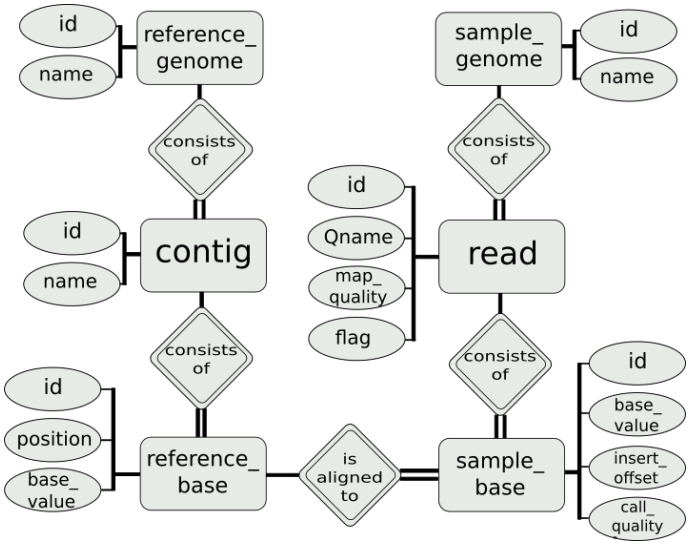

Abbildung 2.3: Datenbankschema für angeordnete Genomdaten [13]

angeordnet sind. Abbildung 2.3 (übernommen nach Dorok et al. [13]) zeigt des Entity-Relationship-Modell des eingesetzten Datenbankschemas. Das Referenzgenom besteht dabei aus einem oder mehreren Contigs, welches wiederum aus mehreren Basen besteht. An jeder Base des Referenzgenomes können mehrere Base des Sample-Genoms angeordnet sein. Jedes dieser Basen gehört zu einem Read. Alle Reads zusammen bilden das Sample-Genom.

Die Attribute *Flag*, *Qname* und *map_ quality* entsprechen den gleichnamigen Feldern des SAM-Formats (vgl. Kap. 2.2). Der *base_ value* einer Base besteht aus einem der Buchstaben 'A', 'C', 'G' oder 'T' und steht für den jeweiligen Typ der Base (vgl. Kap. 2.1). Um eine *Deletion* (vgl. Kap. 2.2 CIGAR-Feld) zu symbolisieren wird eine zusätzliche Pseudobase 'X' eingefügt [12]. Eine *Insertion* zu symbolisieren, besitzt die Base des Sample-Genoms das Attribut *insert_ offset*. Wenn eine Base des Sample-Genoms eine Insertion ist, wird diese zwischen zwei Positionen des Referenzgenoms angeordnet. Um ein wohldefiniertes Schema zu erhalten, wird dieser Base die kleinere der beiden Positionen zugeordnet und der Abstand zu dieser Position als *insert_ offset* hinterlegt [12]. In Listing 2.1 des vorherigen Abschnitts hat die vierte Base des Reads einen *insert_ offset* von 1 und wäre an der Base mit Position 7 angeordnet.

Kapitel 3

Integration von DBMS in die Genomforschung

Um eine Rückwärtskompatibilität zu gewährleisten, muss eine Integration der Datenbanktechnologie in bestehende Workflows der Genomforschung möglich sein. Genomforscher haben im Laufe der Zeit ihre Werkzeuge ausgewählt und Analyse-Pipelines aufgebaut. Es müssen alle wichtigen Programme, Pipelines und Algorithmen trotz eines eingesetzten DBMS als Werkzeug für das Datenmanagement weiterhin funktionieren. Nur so kann gewährleistet werden, dass Genomforscher ohne große Umstellung weiterarbeiten können und eine Chance besteht, dass die Datenbanktechnologie in der Praxis angenommen wird.

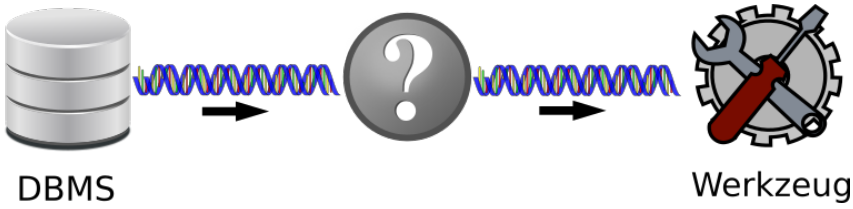

Abbildung 3.1: Wie kann man die Werkzeuge in die Datenbanktechnik integrieren?

Um dieses Ziel zu erreichen, stellt dieses Kapitel zunächst drei mögliche Integrationsansätze vor. Um eine Entscheidungsgrundlage zu schaffen, wird jeder dieser Ansätze anhand der vier Kriterien *Aufwand*, *Daten-Transfer-Overhead*, *Portabilität zwischen DBMS* und *Anwendbarkeit* bewertet. Mithilfe dieser Bewertung wird anschließend der vielversprechendste Integrationsansatz ausgewählt und später am Beispiel von CoGaDB umgesetzt.

3.1 Integrationsansätze

Erweiterung bestehender Werkzeuge um eine Schnittstelle mit einem DBMS Der erste Integrationsansatz besteht darin, bestehende Werkzeuge so anzupassen, dass diese die Daten direkt aus dem DBMS abgreifen. Dabei kann so vorgegangen werden, dass

die benötigten Genomdaten über die SQL-Schnittstelle des DBMS abgefragt und für den weiteren Gebrauch aufbereitet werden. Abbildung 3.2 zeigt, wie dieser Ansatz aussehen könnte. Der Pfeil symbolisiert dabei, an welcher Stelle wir die Integration implementieren müssen. Bei diesem Integrationsansatz muss das Werkzeug um eine SQL-Schnittstelle erweitert werden. Außerdem muss eine Verbindung mit dem DBMS eingerichtet werden.

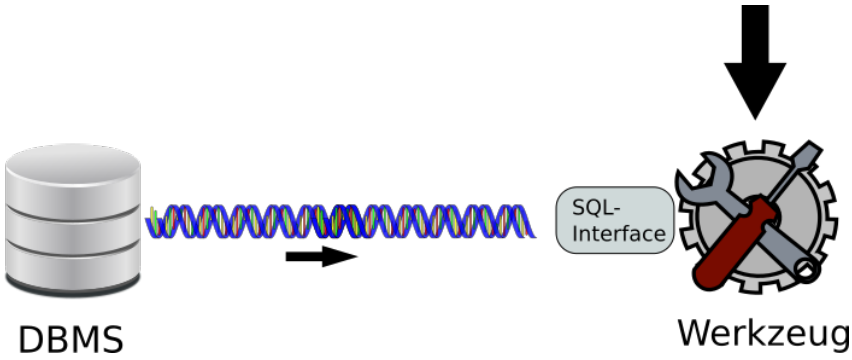

Abbildung 3.2: Erweiterung bestehender Werkzeuge um eine SQL-Schnittstelle

Integration bestehender Werkzeuge in ein DBMS Dieser Integrationsansatz besteht darin, die Werkzeuge und Algorithmen der Genomanalyse direkt in einem DBMS durchzuführen. Dazu werden die Algorithmen und Werkzeuge direkt in den Programmcode des DBMS integriert. Durch einen neuen Custom-Befehl werden die integrierten Werkzeuge anschließend angesprochen. Die Ergebnisse werden direkt im DBMS errechnet und das Ergebnis dargestellt. In Abbildung 3.3 symbolisiert der Pfeil, dass die Integration für diesen Ansatz direkt in das DBMS implementiert werden muss.

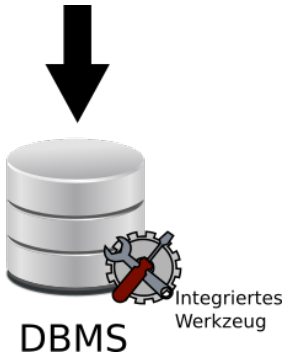

Abbildung 3.3: Integration bestehender Werkzeuge in ein DBMS

Exportfunktion in universelles Dateiformat Der letzte der im Rahmen dieser Arbeit vorgestellten Integrationsansätze besteht darin, die Genomdaten aus dem DBMS in ein weit verbreitetes Dateiformat der Genomforschung umzuwandeln. Die Idee hinter diesem Ansatz ist, dass die meisten Werkzeuge und Algorithmen der Genomforschung diese einfachen Dateiformate als Datengrundlage verwenden. Daher sollten die am meisten ver-

breiteten Datenformate ausgewählt werden, damit so viele Werkzeuge und Algorithmen integriert werden können wie möglich. Weil die Standard Schnittstellen von DBMS und den Werkzeugen nicht zusammenpassen, stellen die Export-Schnittstelle und das Dateiformat eine Art Adapter zwischen den beiden Schnittstellen dar [15]. In Abbildung 3.4 sieht man den Aufbau. Der Benutzer kann den benötigten Bereich angeben. Die Genomdaten dieses Bereichs werden danach in das Dateiformat exportiert und die Datei anschließend von den gewünschten Werkzeugen geladen.

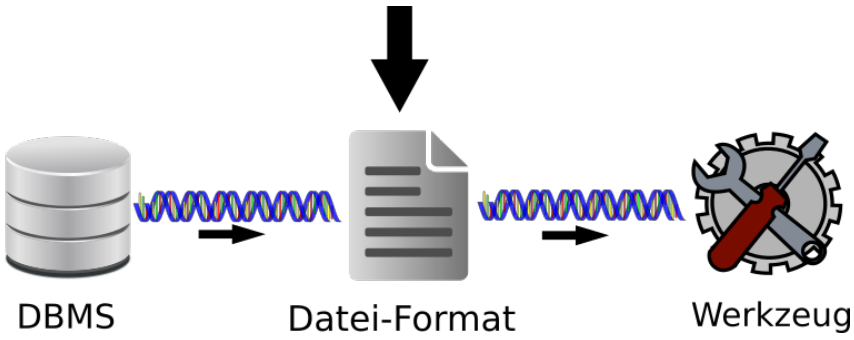

Abbildung 3.4: Exportfunktion in ein universelles Dateiformat

Es kann auf den ersten Blick nicht entschieden werden, welcher der drei vorgestellten Integrationsansätze am vielversprechendsten ist. Daher werden im Folgenden Bewertungskriterien definiert und die Integrationsansätze anhand dieser evaluiert.

3.2 Bewertungskriterien

Bewertungsskala und Gewichtung Um einen konkreten Wert für die Bewertungskriterien zu erhalten, wird eine Bewertungsskala definiert. Diese muss so abstrakt sein, dass sie auf alle Bewertungskriterien anwendbar ist. Außerdem muss nach der Bewertung leicht erkennbar sein, welches der diskutierten Integrationsansätze am geeignetsten ist. Daher bilden wir die Bewertung eines Kriteriums auf eine Zahl zwischen 0 und 1 ab. Je kleiner dabei die Bewertung, desto besser. Weil nicht alle Bewertungskriterien dieselbe Wichtigkeit im Kontext dieser Arbeit besitzen, müssen die Werte für das Endresultat noch gewichtet werden.

Aufwand Der Aufwand eines Integrationsansatzes bemisst sich darin, wie aufwendig es ist, bestehende Werkzeuge in diesen zu integrieren. Er untergliedert sich in Implementationsaufwand und Wartbarkeit. Implementationsaufwand beschreibt den Aufwand, der benötigt wird, bis die gewünschten Werkzeuge und Algorithmen zum ersten Mal in ein DBMS integriert sind. Nach der Implementation beginnt der Aufwand für die Wartbarkeit. Die Wartbarkeit ist die Fähigkeit einer Software, sich nach der Fertigstellung zu verändern und anzupassen [18]. Dabei wird zum Beispiel bewertet, wie aufwendig es ist, Komponenten

nachträglich zu verändern, wie stabil die restlichen Komponenten dabei bleiben und wie aufwendig es ist, die Korrektheit der gesamten Software nach einer Veränderung zu validieren [18]. Für die Bewertungsskala wird der Anteil der Werkzeuge oder DBMS gewählt, der für eine Integration verändert bzw. gewartet werden muss. Wenn für eine Integration beispielsweise jedes Werkzeug der Genomforschung angepasst und gewartet werden muss, so wird der Aufwand mit 1 bewertet. Der Aufwand ist ein wichtiges Bewertungskriterium, weil eine Integration schnell umgesetzt werden muss. Nur so kann der Forschungsstand schnell in die Praxis integriert und somit der praktische Nutzen evaluiert werden. Daher wird dieses Kriterium mit dem Faktor 2 gewichtet.

Daten-Transfer-Overhead Der Daten-Transfer-Overhead bemisst den Overhead, der bei einem Integrationsansatz benötigt wird, um die Daten aus dem DBMS zu extrahieren und für die weitere Verwendung aufzubereiten. Für die Bewertungsskala wird untersucht, welcher Overhead durch das Transferieren der Daten von einem DBMS in das integrierte Werkzeug entsteht. Der Daten-Transfer-Overhead gehört ebenfalls zu den wichtigen Kriterien. Das Ziel, das mit einem DBMS als Datengrundlage verfolgt wird, ist das Datenmanagement zu optimieren. Wenn wir jedoch einen zu hohen Overhead durch den Transfer der Daten bekommen, ist dies nicht möglich. Daher wird dieses Kriterium mit dem Faktor 2 gewichtet.

Portabilität zwischen DBMS Das Kriterium Portabilität bewertet, inwiefern sich der Integrationsansatz auf andere Systeme übertragen lässt. Hoffmann definiert Portabilität von Software durch die Bewertung, wie leicht ein Software-System in eine neue Umgebung übertragen werden kann, also wie abhängig es von der Plattform ist, für die es entwickelt wurde [17]. Im Kontext dieser Arbeit soll bewertet werden, wie abhängig eine Umsetzung der Integrationsansätze von dem DBMS ist, für das es entwickelt wurde. Dabei wird bewertet, welcher Anteil der Werkzeuge für eine Übertragung in ein neues DBMS verändert bzw. neu implementiert werden muss. Die Portabilität ist in der Praxis wichtig, damit Benutzer selbst entscheiden können, welches DBMS sie benutzen. Im Rahmen dieser Arbeit wird jedoch vorrangig am Beispiel eines einzelnen DBMS evaluiert. Daher wird dieses Kriterium nur mit dem Faktor 1 gewichtet.

Anwendbarkeit Die Anwendbarkeit beurteilt, ob alle Werkzeuge und Algorithmen mithilfe des Integrationsansatzes integriert werden können. Dafür werden die zwei Ausprägungsstufen *generelle Anwendbarkeit* und *bedingte Anwendbarkeit* definiert. Für die Bewertungsskala wird der Anteil der Werkzeuge bewertet, auf die der jeweilige Integrationsansatz nicht anwendbar ist.
Ein Integrationsansatz wird als *generell anwendbar* bezeichnet, wenn alle Werkzeuge und Algorithmen mit diesem integriert werden können. Diese Integrationsansätze werden mit

0 bewertet. Ein Integrationsansatz ist *bedingt anwendbar*, wenn es eine Klasse von Werkzeuge oder Algorithmen gibt, die durch diesen nicht direkt integriert werden können. Diese werden mit $\frac{1}{2}$ bewertet, da nur ein Teil der Werkzeuge direkt in dem DBMS umsetzbar ist. Integrationsansätze, die gar nicht anwendbar sind, werden in dieser Arbeit nicht betrachtet. Deshalb gibt es keine Bewertung von 1. Dieses Bewertungskriterium ist nicht ganz so bedeutend im Kontext dieser Arbeit, weil die Integration zunächst nur an ausgewählten Werkzeugen durchgeführt werden soll. Daher wird es mit dem Faktor 1 gewichtet.

3.3 Diskussion

Dieser Abschnitt bewertet die drei vorgestellten Integrationsansätze anhand der vorgestellten Kriterien und wählt anschließend den vielversprechendsten aus.

3.3.1 Aufwand

Erweiterung bestehender Werkzeuge um eine Schnittstelle mit einem DBMS Wenn Werkzeuge oder Algorithmen integriert werden sollen, muss jedes dieser Werkzeuge um eine SQL-Schnittstelle erweitert werden. Bestehende Werkzeuge haben in der Regel keine SQL-Schnittstelle, sondern sind darauf fokussiert, mit einfachen Dateien als Datengrundlage zu arbeiten. Eine Nachrüstung von den vielen kleinen Genomanalyse-Werkzeugen um eine SQL-Schnittstelle ist eine große Herausforderung. Es muss ein standardisiertes Datenbankschema existieren, aus dem alle Werkzeuge die benötigten Daten abfragen können. Eine nachträgliche nicht kompatible Änderung des Schemas zieht weitreichenden Wartungsaufwand mit sich. Die Schnittstelle jedes der integrierten Werkzeuge muss auf das neue Datenbankschema umgestellt werden. Der Aufwand für die Wartbarkeit ist ähnlich groß. Jedes der implementierten SQL-Werkzeuge muss nach der Implementation gewartet werden. Änderungen im Datenbankschema müssen zum Beispiel auf alle Werkzeuge übertragen werden. Wenn ein Werkzeug selbst verändert wird, muss immer sichergestellt werden, dass eine Kompatibilität zu dem implementierten SQL-Interface noch gewährleistet ist. Weil für eine Integration jedes Werkzeug der Genomforschung angepasst und gewartet werden muss, ergibt sich eine Bewertung von 1 für den Aufwand.

Integration bestehender Werkzeuge in ein DBMS Im zweiten Ansatz ist der Aufwand ähnlich wie beim ersten. Es müssen alle Werkzeuge und Algorithmen der Genomforschung neu implementiert und in die Datenbank integriert werden. Die Wartbarkeit ist ebenso wie der Implementationsaufwand sehr hoch. Eine Änderung in der Implementation des DBMS kann dafür sorgen, dass integrierte Werkzeuge oder Algorithmen nicht mehr funktionieren. Durch den Einsatz von automatisierten Komponententests kann man die Auswirkungen einer Änderung auf integrierte Werkzeuge testen. Der Implementationsaufwand wird durch die Bereitstellung der Softwaretests allerdings noch weiter erhöht.

Für eine Integration muss die Funktionalität jedes Werkzeugs der Genomforschung in das DBMS übertragen werden. Bei jeder Änderung im DBMS muss sichergestellt werden, dass alle Werkzeuge danach noch korrekt arbeiten. Daher wird der Aufwand für diesen Integrationsansatz mit 1 bewertet.

Exportfunktion in universelles Dateiformat Der Implementierungsaufwand dieses Integrationsansatzes ist minimal. Das Dateiformat wird so ausgewählt, dass die meisten Werkzeuge und Algorithmen der Genomforschung bereits eine Schnittstelle zu diesem haben. Somit muss lediglich dafür gesorgt werden, dass die Genomdaten aus dem DBMS korrekt und effizient in dieses Dateiformat umgewandelt werden. Die Wartbarkeit ist genauso wie der Implementierungsaufwand sehr gering. Wenn die Datenextraktion auf einer Schnittstelle des eingesetzten DBMS basiert, muss bei Änderungen lediglich dafür gesorgt werden, dass diese Schnittstelle nicht verändert wird. Im Worst Case kann die Änderung an der DBMS-Implementierung dafür sorgen, dass die Genomdaten nicht mehr korrekt in das Dateiformat exportiert werden. Durch automatisierte Softwaretests kann dies jedoch effizient nach jeder Änderung überprüft werden. Weil durch die einmalige Implementation der Export-Schnittstelle alle Werkzeuge und Algorithmen integriert werden können, die das gewählte Dateiformat unterstützen, wird der Aufwand mit 0 bewertet.

3.3.2 Daten-Transfer-Overhead

Erweiterung bestehender Werkzeuge um eine Schnittstelle mit einem DBMS Die Daten werden über das SQL-Interface abgefragt und anschließend in das gewünschte Werkzeug eingelesen. Dadurch ergibt sich ein mittelmäßiger Overhead, weil die Daten nur einmal übertragen werden müssen. Die Bewertung wird daher auf $\frac{1}{2}$ angesetzt.

Integration bestehender Werkzeuge in ein DBMS Weil die Daten in dem DBMS angefragt und direkt verarbeitet werden, entsteht kein Daten-Transfer-Overhead bei diesem Integrationsansatz. Die Bewertung ist daher 0.

Exportfunktion in universelles Dateiformat Der Daten-Transfer-Overhead ist von den hier vorgestellten Integrationsansätzen am höchsten. Die Daten werden zunächst aus der Datenbank abgefragt, in das ausgewählte Dateiformat umgewandelt und in eine Datei geschrieben. Diese Datei wird anschließend in das gewünschte Werkzeug importiert. Weil die Daten zweimal transferiert werden, ergibt sich eine Bewertung von 1.

3.3.3 Portabilität zwischen DBMS

Erweiterung bestehender Werkzeuge um eine Schnittstelle mit einem DBMS Um beim ersten Integrationsansatz eine Implementierung in ein neues DBMS zu übertragen, muss zunächst das Datenbankschema in dem neuen DBMS eingerichtet werden. Weil

jedes DBMS seinen eigenen SQL-Dialekt hat, muss auch überprüft werden, ob die eingesetzten Queries der Werkzeuge mit dem neuen DBMS noch funktionieren. Falls es dabei zu Komplikationen kommt, müssen die Queries entsprechend angepasst werden. Auf jeden Fall muss die Verbindung von jedem der Werkzeuge zu dem neuen DBMS konfiguriert werden. Daher ergibt sich insgesamt eine Bewertung von 1.

Integration bestehender Werkzeuge in ein DBMS Portabilität ist bei diesem Integrationsansatz nicht gegeben. Die Werkzeuge werden in ein konkretes DBMS integriert. Eine Umstellung auf ein anderes DBMS ist gleichbedeutend mit einer Neuimplementation aller integrierten Werkzeuge. Es müssten alle Werkzeuge für den Einsatz eines anderen DBMS neu integriert werden. Daraus ergibt sich eine Bewertung von 1.

Exportfunktion in universelles Dateiformat Die Portabilität wird als sehr gut bewertet. Die DBMS-SAM-Schnittstelle muss zwar für jedes DBMS individuell implementiert werden, jedoch ist eine Anpassung der Werkzeuge nicht nötig, solange dafür gesorgt wird, dass die Datei korrekt exportiert wird. Daher ergibt sich eine Bewertung von 0.

3.3.4 Anwendbarkeit

Erweiterung bestehender Werkzeuge um eine Schnittstelle mit einem DBMS Dieser Integrationsansatz ist nur bedingt anwendbar, weil Werkzeuge, die nicht quelloffen sind, nicht um eine SQL-Schnittstelle erweitert werden können. Dazu müsste man die Entwickler davon überzeugen, dies zu implementieren. Aufgrund der bedingten Anwendbarkeit wird dieser Ansatz mit 1/2 bewertet.

Integration bestehender Werkzeuge in ein DBMS Der zweite Integrationsansatz ist wiederum nur bedingt anwendbar. Nicht alle Werkzeuge und Algorithmen lassen sich direkt in einem DBMS umsetzen. Werkzeuge zur Genomvisualisierung zum Beispiel können nicht direkt in einem DBMS umgesetzt werden. Diese basieren auf GUIs, die auf das DBMS aufgesetzt werden müssten. Auch dieser Integrationsansatz wird aufgrund seiner bedingten Anwendbarkeit mit 1/2 bewertet.

Exportfunktion in universelles Dateiformat Der dritte Integrationsansatz ist generell anwendbar. Alle wichtigen Werkzeuge und Algorithmen arbeiten mit einem Dateiformat als Datengrundlage. Wegen seiner generellen Anwendbarkeit wird dieser Ansatz mit 0 bewertet.

Bewertungs-kriterium (Gewichtung)	**Anpassen der Werkzeuge**	**Integration der Werkzeuge**	**Exportfunktion**
Aufwand (2x)	1	1	0
Daten-Transfer-Overhead (2x)	1	0	1
Portabilität zwischen DBMS (1x)	1	1	0
Anwendbarkeit (1x)	$\frac{1}{2}$	$\frac{1}{2}$	0
Gesamt mit Gewichtung	$5\frac{1}{2}$	$3\frac{1}{2}$	2

Tabelle 3.1: Bewertung der vorgestellten Integrationsansätze

3.4 Zusammenfassung und Entscheidung

Im ersten Integrationsansatz werden bestehende Werkzeuge der Genomforschung um eine Schnittstelle zu einer DBMS erweitert. Der Aufwand und die Portabilität sind als sehr hoch einzustufen, da jedes Werkzeug angepasst werden muss. Die Anwendbarkeit ist nur bedingt, weil Werkzeuge, die nicht quelloffen sind, nicht angepasst werden können. Auch müssen alle Daten aus dem DBMS in das Werkzeug transferiert werden, wodurch ein hoher Daten-Transfer-Overhead entsteht. Insgesamt werden bei diesem Integrationsansatz viele Bewertungskriterien als hoch eingestuft und daher ist dieser Ansatz mit einer Gesamtbewertung von $4\frac{1}{2}$ der am wenigsten vielversprechendste der hier vorgestellten. Der zweite Integrationsansatz besteht darin, alle bestehenden Werkzeuge direkt in den Programmcode eines DBMS zu integrieren. Bei diesem Ansatz existiert kein Datentransfer-Overhead, da die Daten direkt in dem DBMS verarbeitet werden. Allerdings müssen auch hier Aufwand und Portabilität zwischen DBMS als sehr hoch bewertet werden, weil jedes Werkzeug in den Quellcode jedes DBMS integriert werden muss. Außerdem können zum Beispiel Werkzeuge, die auf einer GUI basieren, nicht direkt in das DBMS integriert werden, wodurch dieser Ansatz nur bedingt anwendbar ist. Die Gesamtbewertung dieses Integrationsansatzes inklusive Gewichtung ist $3\frac{1}{2}$ und er steht damit an zweiter Position.
Der dritte Integrationsansatz besteht darin, den gewünschten Bereich der Genomdaten in ein universelles Dateiformat der Genomforschung zu exportieren. Der Aufwand und die Portabilität dieses Integrationsansatzes sind minimal, weil die Schnittstelle zentral Implementiert werden kann und damit alle Werkzeuge und Algorithmen direkt angebunden sind. Weil der Einsatz von einfachen Dateien als Datengrundlage gegenwärtiger Standard ist, gibt es keine Klasse von Werkzeugen, die nicht unterstützt werden. Einzig der Daten-Transfer-Overhead ist sehr hoch, weil die Daten zweimal transferiert werden müssen. Einmal von

dem DBMS in die Datei und anschließend nochmal von der Datei in das Werkzeug. In der Praxis werden meistens jedoch nur Teile eines Genoms zur selben Zeit analysiert bzw. verarbeitet. Daher ist das zu exportierende Datenvolumen oft nicht sehr hoch und damit der hohe Daten-Transfer-Overhead annehmbar. Nach Gewichtung ergibt sich eine Gesamtbewertung von 2.

Tabelle 3.1 zeigt eine Gesamtübersicht über die Bewertung der drei Integrationsansätze. Aus der Bewertung ergibt sich, dass der dritte Integrationsansatz, der die Anbindung durch eine Exportfunktion in ein universelles Dateiformat der Genomforschung löst, am vielversprechendsten ist. Daher wird dieser Ansatz im folgenden Kapitel zunächst konzeptionell umgesetzt und anschließend am Beispiel von CoGaDB und IGV implementiert. In Kapitel 5 wird der Daten-Transfer-Overhead dieses Ansatzes an einer konkreten Implementation noch einmal im Detail evaluiert.

Kapitel 4

DBMS-SAM-Schnittstelle

Dieses Kapitel stellt eine Umsetzung des im Kapitel 3.1 beschriebenen Integrationsansatz mittels Exportfunktion in ein universelles Dateiformat vor. Als Dateiformat wird hier das SAM-Format [39] verwendet, weil dieses für *aligned* Genomdaten weit verbreitet ist. Um Genomdaten in einem DBMS zu speichern, wird das in Kapitel 2.4 vorgestellte Genomdatenmodell nach Dorok et al. [13] verwendet. Zwischen diesem Modell und dem SAM-Format bestehen grundsätzliche Unterschiede im Ansatz zur Speicherung von Genomdaten. Der bedeutendste Unterschied besteht darin, dass im SAM-Format für jedes Read eine Zeile erzeugt wird und die Genomdaten demnach Read-weise gruppiert sind. Das Genomdatenmodell sieht vor, dass pro Base ein Datenbankeintrag erzeugt wird, die Genomdaten also basenweise gruppiert sind. Außerdem ist kein direktes Feld im Genomdatenmodell für den Cigar-String vorgesehen und dieser muss aus anderen Daten errechnet werden. Daher wird in diesem Kapitel ein Modul konzeptioniert, das Genomdaten eines relationalen DBMS in das Dateiformat SAM exportiert. Zunächst wird konzeptionell vorgestellt, wie die Daten aus einem DBMS gewonnen und anschließend in das SAM-Format umgewandelt werden. Anschließend wird eine konkrete Implementierung dieses Ansatzes am Beispiel von CoGaDB vorgestellt, das an das Werkzeug IGV angebunden wird.

4.1 Konzept

In diesem Abschnitt wird ein Ansatz vorgestellt, der beschreibt, wie die für das SAM-Format relevanten Genomdaten aus einem DBMS gewonnen und anschließend in das SAM-Format umgewandelt werden können. Dazu werden zunächst die Anforderungen einer SAM-Schnittstelle herausgestellt und ein genereller Ansatz entwickelt. Anschließend werden mögliche Ansätze der Datenextraktion diskutiert. Der Abschnitt schließt mit einem Konzept zur Konvertierung der gewonnenen Daten aus dem DBMS in das SAM-Format.

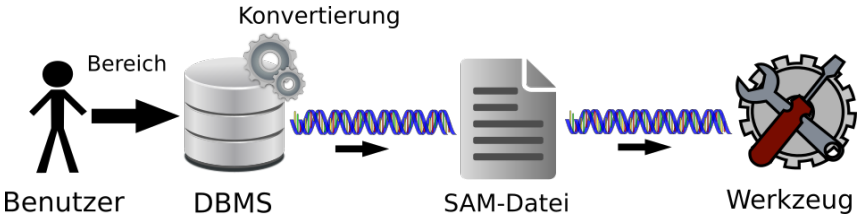

Abbildung 4.1: DBMS-SAM-Schnittstelle Überblick

4.1.1 Überblick

Genereller Ablauf Die DBMS-SAM-Schnittstelle stellt eine Verbindung zwischen einem DBMS und bestehenden Werkzeugen der Genomforschung her. Der generelle Ablauf gleicht einer Pipeline, die in Abbildung 4.1 dargestellt ist. Der Benutzer kann zunächst den benötigten Bereich auswählen. Dieser Bereich wird durch die Angabe von ContigIDs sowie einer minimalen und maximalen Position festgelegt. Die Genomdaten aus dem DBMS, die sich in diesem definierten Bereich befinden, werden extrahiert und direkt im DBMS in das SAM-Format konvertiert und anschließend in eine SAM-Datei geschrieben. Der Dateiname kann vom Benutzer frei gewählt werden, die Dateiendung ist jedoch auf *.sam* festgesetzt. Außerdem wird eine Werkzeugsteuerung direkt in die Schnittstelle implementiert. Diese kommuniziert direkt mit den gewünschten Werkzeugen und gibt diesen die Befehle zum Laden der exportierten SAM-Datei. Zur späteren Verifikation der DBMS-SAM-Schnittstelle wird die Zeit eines Export-Durchlaufs gemessen, wenn die Zeitmessung eingeschaltet ist.

4.1.2 Datengewinnung

Da der Benutzer den Bereich definieren kann, der exportiert werden soll, und die Datenbank auch für die SAM-Datei nicht relevante Daten enthält, müssen die benötigten Daten zunächst aus der Datenbank extrahiert werden. In diesem Abschnitt werden zwei Ansätze der Datengewinnung vorgestellt. Zum einen ist es möglich, direkt auf der physikalischen Ebene des DBMS zu agieren und einen Algorithmus zu schreiben, der die Daten manuell zusammensucht. Ein zweiter Ansatz arbeitet auf einer höheren Ebene. Die Daten werden dabei zunächst durch SQL-Queries vorgefiltert, um anschließend die Datenkonvertierung auf dem Ergebnis dieser Queries durchzuführen. Während der zweite Ansatz die internen Mechanismen des DBMS ausnutzt, muss im ersten Ansatz selbst dafür gesorgt werden, dass die Daten effizient und vollständig zusammengetragen werden. Ein selbst entwickelter Algorithmus, der auf der physikalischen Ebene des DBMS arbeitet, ist in der Regel fehleranfälliger und auch weniger effizient als ein Ansatz, der bereits erprobte Mechanismen eines DBMS benutzt, um die Daten zusammenzutragen. Aus diesem Grund wird im Folgenden ausschließlich auf den zweiten Ansatz eingegangen.

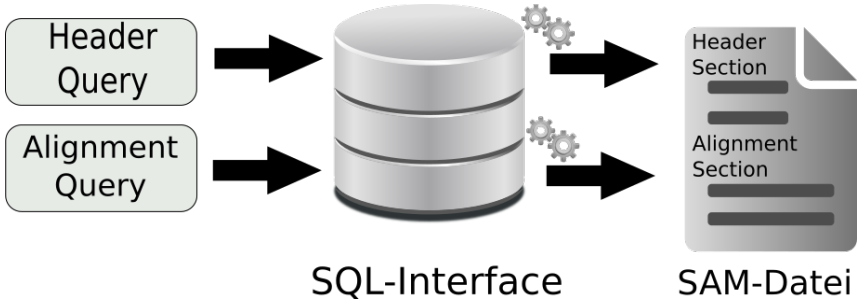

Abbildung 4.2: Datenextraktion auf Basis von SQL-Queries

Daten zur Header-Generierung Um die SQ-Zeilen der Header Section zu generieren, wird der c_name Wert jedes Contigs des zu exportierenden Bereichs verwendet, um die SN-Tags zu generieren. Die LN-Tags werden aus der Anzahl der Basen des jeweiligen Contigs erstellt. Mithilfe der SQL Query in Listing 4.1 können alle zum Erstellen des Headers benötigten Daten aus der Datenbank extrahiert werden. Im Folgenden soll diese Query als *Header Query* bezeichnet werden. Die Where-Klausel wird aus der Bereichseinschränkung der Benutzereingabe durch eine Disjunktion aller angegebenen Contig-IDs dynamisch erzeugt. Die Where-Klausel aus der SQL Query in Listing 4.1 wird demnach erzeugt, wenn die beiden Contigs mit den IDs 2344 und 23023 exportiert werden sollen.

```
1 select c_id, c_name, count(rb_id)
2 from reference_base join contig on rb_c_id=c_id
3 where c_id=2344 or c_id=23023
4 group by c_id, c_name
5 order by c_id;
```

Listing 4.1: Header Query

Daten zur Alignment-Generierung Die Felder QNAME, FLAG, RNAME und MAPQ der SAM-Datei werden direkt aus den Datenbankfeldern r_qname, r_flag, c_name und r_mapq der Tabellen Read und Contig übernommen. Die Felder TLEN, POS, SEQ und QUAL der SAM-Datei müssen aus den Werten der Datenbankfelder rb_position, sb_base_value und sb_base_call_quality berechnet werden. Die Berechnung des CIGAR-Strings erfolgt durch die Werte der beiden Datenbankfelder sb_base_value und sb_insert_offset. Weil Beziehungen zwischen Reads zum Zeitpunkt dieser Arbeit noch nicht vollständig unterstützt werden, wird für die Felder RNEXT und PNEXT der Default-Wert verwendet. Daher müssen für diese Felder keine zusätzlichen Daten aus dem DBMS extrahiert werden. Damit die Ergebnisse der Extraktion in der richtigen Reihenfolge sind, müssen diese nach contig_id, sb_read_id, rb_position, sb_insert_offset sortiert werden. Die Sortierung nach contig_id und sb_read_id sorgt dafür, dass zusammengehörige Daten eines Reads nicht vermischt werden. Durch die Sortierung nach rb_position und sb_insert_offset wird

sichergestellt, dass die Reads in der richtigen Reihenfolge zusammengesetzt werden. Die SQL Query aus Listing 4.2 liefert alle für die Berechnung der Alignment Section erforderlichen Daten. Im Folgenden soll diese Query als *Alignment Query* bezeichnet werden. Die Where-Klausel wird wie zuvor dynamisch aus der Bereichseinschränkung der Benutzereingabe erzeugt, indem der rb_position-Bereich durch die angegebene minimale und maximale Position eingeschränkt wird. Außerdem wird c_id durch die vom Benutzer angegebenen contig-IDs eingeschränkt. Die Where-Klausel aus Listing 4.2 wird beispielsweise erzeugt, wenn das Contig mit der ID 234 von Position 100 bis zu Position 500 exportiert werden soll.

```
1 select sb_read_id, r_qname, r_flag, c_name,
2 rb_position, r_mapq,
3 sb_base_value, sb_insert_offset,
4 sb_base_call_quality
5 from sample_base
6 join reference_base on sb_rb_id = rb_id
7 join contig on rb_c_id = c_id
8 join read on sb_read_id = r_id
9 where c_id=234 and rb_position between 100 and 500
10 order by c_id, sb_read_id, rb_position, sb_insert_offset;
```

Listing 4.2: Alignment Query

Ausblick Ein möglicher dritter Ansatz verbindet die Datengewinnung mit Teilen der Datenkonvertiertung. Dabei wird das SQL-Interface eines DBMS um mehrere benutzerdefinierte Aggregatfunktion erweitert. So ist beispielsweise eine Aggregatfunktion denkbar, welche die sb_base_value-Werte sequenziell zusammensetzt und die sb_base_call_quality in Character umrechnet und anschließend zusammensetzt. Eine weitere Aggregatfunktion kann den Cigarstring aus sb_base_value und sb_insert_offset berechnen. Durch diese Aggregatfunktionen wäre es möglich, den Alignment-Bereich der SAM-Datei durch eine einzige SQL Query berechnen zu lassen. Benötigte Werte, die sich nicht direkt in dem DBMS befinden, können durch die beschriebenen Aggregatfunktionen errechnet werden. Weil die Möglichkeiten und Funktionsweisen von benutzerdefinierten Aggregatfunktionen jedoch stark von dem eingesetzten DBMS abhängen und hier zunächst ein grundsätzlicher Ansatz vorgestellt werden soll, wird dies nicht Teil der vorliegenden Arbeit sein.

4.1.3 Datenkonvertierung

Weil das Ergebnis der SQL-Queries nicht alle benötigten Daten des SAM-Formats direkt enthält und auch die richtige Anordnung nicht gegeben ist, müssen die Daten konvertiert und richtig angeordnet werden, bevor sie in eine Datei geschrieben werden.

Berechnung der Header-Zeilen Der Header der SAM-Datei ist optional, soll jedoch zur Steigerung der Kompatibilität zu externen Genom-Programmen mit exportiert werden. Als erste Zeile des Headers wird die Default-Zeile *@HD VN:1.5 SO:coordinate* verwendet. Diese besagt, dass die Datei der Version 1.5 der SAM-Spezifikation entspricht [39]. Außerdem wird durch das Setzen des SO-Tags zu coordinate angegeben, dass der Wert von RNAME zur Sortierung verwendet wird. Als Grundlage wird die Reihenfolge benutzt, die durch die @SQ-Zeilen impliziert wird [39]. Diese Sortierung ist gegeben, weil wir sowohl in der Header Query als auch in der Alignment Query primär nach dem Attribut c_id sortieren.
Die @SQ-Zeilen werden dynamisch erzeugt, indem eine Schleife über das Ergebnis der Header Query iteriert. Da die Header Query nach Contig gruppiert, kann für jeden Eintrag der Ergebnis-Tabelle eine neue @SQ-Zeile erzeugt werden. Der SN-Tag einer @SQ-Zeile kann durch den Wert von c_name gesetzt werden. Der LN-Tag ergibt sich aus dem Wert von count(r_id).

Berechnung der Alignment-Zeilen In der Alignment Query ist es im Allgemeinen nicht möglich, nach Reads zu gruppieren. Dazu müssten im DBMS spezielle Aggregatfunktionen implementiert sein, welche die Felder CIGAR, SEQ und QUAL aus den einzelnen Basenwerten errechnen. Wie jedoch zuvor erwähnt, soll dieser Ansatz nicht Teil dieser Arbeit sein. Daher extrahiert die Alignment Query die Daten basenweise aus dem DBMS. Die Query übernimmt die Sortierung sowie die Auswahl der Daten. Durch eine Iteration mit einer Schleife über das Ergebnis der Query kann nun die SAM-Datei erzeugt werden. Die Werte der Felder QNAME, FLAG, RNAME und MAPQ können wie oben bereits beschrieben direkt aus der Datenbank abgelesen werden. Die Felder PNEXT und RNEXT werden auf Ihre Default-Werte gesetzt. Der Default-Wert ist für PNEXT gleich '0' und für RNEXT gleich '*'.

Berechnung des Cigar Strings Der Cigar-String wird nicht direkt in dem Genomdatenmodell hinterlegt und muss sequenziell aus den Werten sb_insert_offset und sb_base_value errechnet werden. Weil nur aligned Basen in dem Modell unterstützt werden, reduzieren sich die relevanten Symbole auf Match (M), Deletion (D) und Insertion (I). Auch die Aufteilung der matched Basen in sequence match (=) und sequence unmatch (X) aus der SAM-Format-Spezifikation soll zunächst nicht unterstützt werden, weil diese Unterteilung in der Praxis nur selten verwendet wird. Berechnet wird der Cigar-String eines Reads mithilfe des endlichen Automaten in Abbildung 4.3. Der Automat besteht aus fünf Zuständen, einem Zähler und der String-Repräsentation des derzeitigen Cigar-Strings. In der Initialisierung wird der Zähler auf 0, die Repräsentation für den Cigar-String auf den leeren String und der Automat auf den Startzustand (S) gesetzt. Die Zustände Match (M), Deletion (D) und Insertion (I) repräsentieren die drei unterstützen Typen der Subsequenzen

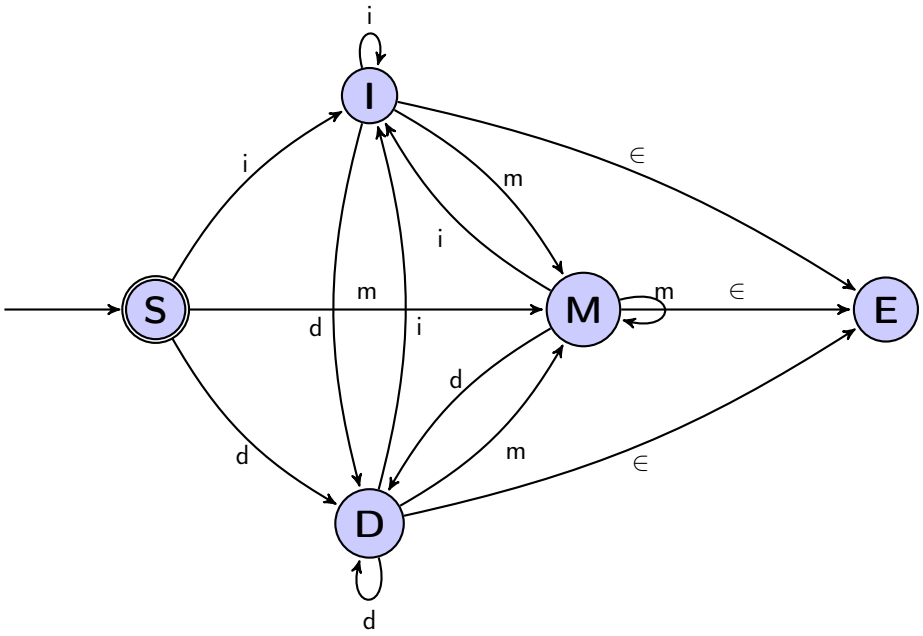

Abbildung 4.3: Cigar-Automat

des Cigar-Strings. Befindet sich der Zustand in einem dieser drei Zustände, bedeutet ein Zustandsübergang in denselben Zustand, dass der Zähler um 1 erhöht wird. Ein Zustandswechsel in einen beliebigen anderen Zustand bedeutet, dass eine neue Subsequenz vom Typ des ausgehenden Zustandes hinten an die String-Repräsentation des Cigar-Strings angehängt wird. Diese hat den Wert des Zählers als Prefix. Außerdem wird noch der Zähler auf 0 resettet. Der Endzustand (E) repräsentiert das Ende der Cigar-String-Berechnung für den gegenwärtigen Read und darf erst eingenommen werden, wenn der letzte Input des derzeitigen Reads verarbeitet wurde. Weil der Übergang in den Endzustand nicht durch ein spezielles Input-Signal, sondern durch das Ende der Sequenz eines Reads signalisiert wird, ist ein Epsilon-Übergang gewählt worden. In einer konkreten Implementierung repräsentiert jedoch der Wechsel der Read-ID, dass die Cigar-String-Berechnung für diesen Read abgeschlossen ist und der Automat in den Endzustand wechseln muss.

Als Input verarbeitet der Automat die Werte von sb_insert_offset und sb_base_value der einzelnen Basen eines Reads. Ein sb_base_value mit dem Wert 'X' bedeutet, dass an dieser Stelle eine Deletion vorliegt. In dem Automaten wird dies mit dem Eingabesymbol 'd' repräsentiert. Ein sb_base_value, der nicht 'X' ist, in Kombination mit einem sb_insert_offset von 0 repräsentiert einen Match. Dies wird durch das Eingabesymbol 'm' repräsentiert. Wenn sb_base_value ungleich 'X' ist und sb_insert_offset einen Wert größer als 0 hat, befindet sich an dieser Stelle eine Insertion. Dies wird durch das Eingabesymbol 'i' repräsentiert. Bei einem Wechsel zum nächsten Read macht der Automat einen Zustandsübergang in den Endzustand. Der dadurch entstandene Cigar-String entspricht dem endgültigen Cigar-String des derzeitigen Reads.

Berechnung der QUAL-, SEQ- und TLEN-Werte Die Qual-Werte der Sam-Datei werden durch sukzessives Zusammensetzen der Werte von sb_base_value der derzeitigen Iteration innerhalb eines Reads erzeugt. Die SEQ-Werte werden, wie in Kapitel 2 beschrieben, in dem Genomdatenmodell als *Phred Quality Score* [2] pro Basis gespeichert und müssen für die SAM-Datei zunächst anhand der ASCII-Tabelle in ein Zeichen umgerechnet und anschließend zusammengesetzt werden. Damit sich diese Zeichen in einem für Menschen sichtbaren Bereich befinden, muss zuvor 33 addiert werden. Durch diese Addition befindet sich der Wert innerhalb der druckbaren Zeichen in der ASCII-Tabelle. Der TLEN-Wert wird errechnet, indem der minimale Positionswert des derzeitigen Reads von dem maximalen subtrahiert und anschließend 1 addiert wird.

4.2 Implementierung am Beispiel von CoGaDB und IGV

Der folgende Abschnitt schlägt eine Implementierung der in Kapitel 4.1 beschriebenen konzeptionellen Umsetzung einer DBMS-SAM-Schnittstelle vor. Als zugrunde legendes Setup der beispielhaften Implementation wird das DBMS CoGaDB [3] gewählt. Um die Integration an ein Werkzeug der Genomforschung zu demonstrieren, wird eine Anbindung an den Integrative Genomics Viewer (IGV) [34] realisiert, die den IGV extern steuert und diesem den Befehl gibt, die exportierte SAM-Datei zu laden und somit zu visualisieren.
Zunächst wird ein grundlegender Überblick über CoGaDB und IGV gegeben und erläutert, wieso die Wahl auf dieses DBMS bzw. dieses Werkzeug fiel. Anschließend wird ein genereller Überblick über die Struktur und die Zusammenhänge des CoGaDB-SAM-Moduls gegeben. Der Abschnitt schließt mit einer detaillierteren Erklärung der einzelnen Untermodule. Dabei werden wichtige Funktionen und Klassen beschrieben. Durch einen Rückbezug auf die Elemente des konzeptionellen Ansatzes wird die Bedeutung der Untermodule näher erläutert.

4.2.1 Setup der Implementation

CoGaDB CoGaDB ist ein spaltenorientiertes DBMS mit GPU-Beschleunigung [3], das von der TU Dortmund und der Universität Magdeburg entwickelt wird[1]. Für CoGaDB existiert eine Erweiterung, die Hilfsmittel und Optimierungen für Genomforschung in CoGaDB zur Verfügung stellt[2]. Das Genomdatenmodell aus Kapitel 2.4 nach Dorok et al. [13] ist bereits in der Genomerweiterung von CoGaDB integriert. Für die beispielhafte Umsetzung des Integrationsansatzes wurde das DBMS CoGaDB ausgewählt, weil dieses auf moderner Technik basiert und moderne Beschleunigungsmechanismen integriert sind und so die effiziente Verwaltung von Genomdaten möglich ist. Weil die Daten im Hauptspeicher liegen, ist der Datenzugriff deutlich effizienter als wenn die Daten auf der Festplatte

[1] http://cogadb.cs.tu-dortmund.de/wordpress/
[2] http://cogadb.cs.tu-dortmund.de/wordpress/current-projects

liegen. Auch kann die Performance des Datenzugriffs durch ausnutzen des Caches weiter verbessert werden. Durch das spaltenorientierte Datenbankdesign ist der Nutzen des Caches effizienter als bei einem zeilenorientierten Datenbankdesign. Außerdem können bessere Kompressionsverfahren eingesetzt werden [12]. CoGaDB wird es unter der *GPL v3-Lizenz*[3] veröffentlicht und kann daher beliebig auf die eigenen Bedürfnisse angepasst werden.

Integrative Genomics Viewer Viele Schritte der Genomforschung können automatisiert durchgeführt werden. Allerdings sind die Interpretation und das Urteil der Genomforscher unumgänglich, um Genomdaten zu untersuchen. Daher ist es besonders wichtig, dass es eine Möglichkeit gibt, Genomdaten zu visualisieren [40]. Aus diesen Gründen wurde ein Tool zur Genomvisualisierung ausgewählt, um den Integrationsansatz exemplarisch zu implementieren. Es existieren viele Werkzeuge zur Genomvisualisierung, wie beispielsweise *Integrated Genome Browser* [28], *Tablet* [26] oder *Bamview* [6]. Für diese Arbeit wurde der IGV als Genomvisualisierungswerkzeug ausgewählt. Der IGV ist ein leichtgewichtiges Werkzeug, um große Datensätze der Genomforschung in Echtzeit auf normalen Desktop-Computern zu visualisieren [34]. IGV ist in der Programmiersprache Java implementiert und funktioniert auf allen großen Betriebssystemen [40]. Die Wahl fiel auf den IGV, weil dieser in der Praxis sehr weit verbreitet ist und viele Dateiformate unterstützt. Außerdem verfügt der IGV über mehrere Schnittstellen, die es ermöglichen, ihn von externen Programmen aus zu steuern [40]. IGV ist frei verfügbar unter der *GNU LGPL-Lizenz*[4].

4.2.2 Überblick über die Implementation

Die Implementation wird in Form eines CoGaDB-SAM-Moduls umgesetzt, das direkt in den Programmcode von CoGaDB integriert wird. Das Modul setzt sich aus fünf Untermodulen zusammen und nutzt zusätzlich die interne SQL-Schnittstelle von CoGaDB. Das CoGaDB-SAM-Modul wird direkt aus dem CoGaDB-Client durch den Befehl *export_sample_genome* aufgerufen.

[3] http://www.gnu.org/licenses/gpl.txt
[4] http://opensource.org/licenses/lgpl-2.1.php

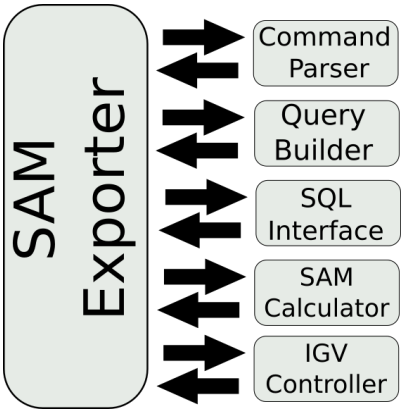

Abbildung 4.4: Überblick CoGaDB-SAM-Modul

Das Untermodul *SAM Exporter* dient als Hauptcontroller, der den Ablauf des CoGaDB-SAM-Moduls auf der höchsten Ebene steuert. Dieses Untermodul wird durch den *export_sample_genome* gestartet und die Argumente diesem übergeben. Die Argumente werden an den jeweils aktivierten *CommandParser* weitergeleitet und von diesem verarbeitet. Anhand von dessen Informationen werden anschließend vom *Query Builder* die benötigte Alignment und Header Query erzeugt. Diese Queries werden auf der internen SQL-Schnittstelle von CoGaDB ausgeführt. Das Ergebnis dieser Query wird an das Untermodul *SAM Calculator* weitergereicht, das die SAM-Datei nach dem in Kapitel 4.1.3 beschriebenen Konzept erzeugt. Wenn die IGV-Steuerung aktiviert ist, wird zum Schluss das Untermodul *IGV-Controller* gestartet, das Befehle zum Laden und Anzeigen der exportierten SAM-Datei sendet.

Konfigurationsmöglichkeiten Das CoGaDB-SAM-Modul kann an zwei Stellen konfiguriert werden. Zum einen sind in dem Variablen-Manager von CoGaDB die fünf in Tabelle 4.1 beschriebenen Variablen hinterlegt. Hierbei handelt es sich um übergreifende Einstellungen, die im Voraus im *Startup-Script* von CoGaDB gesetzt werden können und allgemein gültig sind.

Konfigurationen, die für jeden Aufruf des CoGaDB-SAM-Moduls individuell gesetzt werden müssen, werden direkt als Argumente übergeben. Darunter fällt beispielsweise die Auswahl des zu exportierenden Bereichs oder das Aktivieren von zusätzlichen Funktionen wie der Zeitmessung bzw. IGV-Steuerung.

4.2.3 Command Parser

Das Untermodul Command Parser wird durch eine abstrakte Klasse umgesetzt. Diese dient als Schnittstelle zwischen den Benutzereingaben und dem Query Builder. Durch die virtuelle Funktion addArguments, die von jeder Unterklasse implementiert werden muss, werden die Benutzereingaben verarbeitet. Es wird überprüft, ob diese Eingaben valide sind und der

Variablenname	Erklärung	Default-Wert
sam_save_path	Definiert, wo die exportierte SAM-Datei gespeichert wird	/tmp
sam_exporter_input_api	Definiert, welcher Command-Parser verwendet wird	simple
igv_port	Definiert den Port, über den IGV gesteuert wird	60151
igv_snapshot_path	Definiert, wo Bildschirmfotos des IGV gespeichert werden	/tmp
igv_snapshot_name	Definiert Namen und Format des Bildschirmfotos vom IGV	snapshot.jpg

Tabelle 4.1: Konfigurationsvariablen des CoGaDB-SAM-Moduls

jeweiligen Syntax des Command Parser entsprechen. Anschließend werden die Benutzereingaben in die Attribute der Klasse eingepflegt. So repräsentiert ein Objekt dieser Klasse in Kombination mit den Werten der in Tabelle 4.1 gezeigten Variablen des Variablenmanagers die Gesamtkonfiguration des derzeit ausgeführten Durchlaufs. Andere Klassen und Funktionen des CoGaDB-SAM-Moduls können jederzeit auf die Konfiguration zugreifen. Durch diese Klassenstruktur entsteht eine sehr dynamische Struktur, in der jederzeit ein neuer SAM-Export-Parser hinzugefügt und aktiviert werden kann. Jeder dieser Command Parser interpretiert einen eigenen Aufbau des Parameter-Strings, ohne dass die innere Struktur jedes Mal angepasst werden muss. Welcher Command Parser verwendet wird, kann durch die Variable sam_exporter_input_api des Variablen-Managers definiert werden.
Den Kern des Command Parsers bildet die Auswahl des zu exportierenden Bereichs. Diese wird in dem Attribut SamSelection gespeichert, dessen Typ ein Vector von Integer-Triplets ist. Jedes dieser Triplets repräsentiert dabei den ausgewählten Bereich eines Contig und hat die Form (<ContigId>, <start>, <end>). Die SamSelection wird im Folgenden an das Untermodul Query Builder übergeben und von diesem verwendet, um die Where-Klausel der SQL-Queries zu generieren.
Außerdem können Parameter definiert werden, die von dem Command Parser verarbeitet werden. Durch diese Parametrisierung können weitere Zusatzfunktionen wie das Messen der Laufzeit oder das Steuern von Werkzeugen eingeschaltet werden. Jede dieser Zusatzfunktionen wird durch ein nach außen hin nur lesbares Boolesches Attribut der Klasse repräsentiert. So können die anderen Untermodule des CoGaDB-SAM-Moduls jederzeit abfragen, ob diese Zusatzfunktion aktiviert wurde oder nicht. In einer Command Map werden private Funktionen zum Setzen dieser Attribute mit Parametern verknüpft, die der Benutzer als Argument des Command export_sample_genome übergeben kann. Dieser Ansatz erlaubt es, Parameter in beliebiger Reihenfolge zu akzeptieren. Außerdem können

Parameter	Bedeutung
-t	Aktiviert die Zeitmessung
-i	Aktiviert IGV-Steuerung
-s	Erstellt ein Bildschirmfoto vom IGV

Tabelle 4.2: Valide Parameter des SimpleSamExportParsers

auch schnell und einfach neue Parameter hinzugefügt werden.
Eine für diese Arbeit implementierte Unterklasse ist der SimpleSamExportParser, der das Exportieren eines konkreten Bereichs aus einem oder mehreren Contigs unterstützt. Ein vom SimpleSamExportParser als valide angesehener Argument-String hat die Form <contigID_1>,<contigID_2>,...,<contigID_n> <start> <end> (-<param_1> ... -<param_n>) (<Filename>). Die Angabe von Parametern sowie des Speichernamens ist dabei optional. Wird kein Speichername angegeben, so wird der Default-Name *default.sam* verwendet. Tabelle 4.2 beschreibt die validen Parameter, die im SimpleSamExportParser hinterlegt sind. Anstatt konkreter Werte für *ContigID, Start oder End* kann auch der Spezialwert '*' gesetzt werden. Im Kontext von *ContigID* bedeutet der Spezialwert, dass alle Contigs exportiert werden. Im Kontext von *Start* oder *End* bedeutet der Spezialwert, dass der kleinstmögliche bzw. größtmögliche Wert genommen wird. Mit dem Befehl *export_sample_genome * * **
kann zum Beispiel die gesamte Datenbank exportiert werden.
Weitere Command Parser, mit der auch komplexere Bereiche exportiert werden können, sind denkbar. Da im Kontext dieser Arbeit nur eine Beispielimplementierung angegeben werden soll, um den Integrationsansatz im Allgemeinen zu validieren, wurden keine weiteren Command Parser umgesetzt.

4.2.4 Query Builder und SQL Interface

Query Builder Das Untermodul Query Builder wird durch eine Klasse umgesetzt, in der das Attribut SamSelection des Command Parser im Konstruktor übergeben wird. Aus diesen Informationen wird die Where-Klausel der beiden Queries dynamisch nach dem Konzept in Kapitel 4.1.2 erzeugt.

SQL Interface Um die SQL Queries auszuführen, wird die interne SQL-Schnittstelle von CoGaDB verwendet. An diese werden die Queries als String übergeben. Diese werden von CoGaDB ausgeführt und ein Zeiger auf die Ergebnistabellen zurückgeliefert. Über diese Zeiger ist ein Zugriff auf die komprimierten Daten möglich.

4.2.5 SAM Calculator

Das Untermodul SAM Calculator wird durch die Funktion calculateSamfile(TablePtr, TablePtr, ClientPtr) umgesetzt. Diese ist für die Umrechnung der Daten in das SAM-Format zuständig. Als Parameter bekommt sie zwei TablePtr und einen ClientPtr übergeben. Der erste TablePtr zeigt auf die Ergebnistabelle der Header Query und der zweite TablePtr auf die Ergebnistabelle der Alignment Query. Der ClientPtr dient ausschließlich zum Auslesen des Output-Streams des Clients. Die Errechnung der Header-Zeile ist eine direkte Umsetzung des in Kapitel 4.1.3 beschriebenen Konzepts und daher trivial. Abbildung 4.5 gibt einen Überblick über die Berechnung der Alignment-Zeilen anhand eines UML-Aktivitätsdiagramms[5]. Zuerst werden die ColumnPointer der Tabelle typisiert. Dadurch muss der enthaltene Wert nicht bei jedem Zugriff gecastet werden, wodurch die Performance deutlich verbessert wird. Für jedes Feld des SAM-Formats wird eine Variable initialisiert. Dann beginnt eine Schleife, die über die Zeilen der Ergebnistabelle iteriert. In jeder Schleifeniteration wird erst überprüft, ob ein neuer Read begonnen hat, was durch den Wechsel der Read-ID symbolisiert wird. Ist dies der Fall, werden die errechneten Felder des letzten Reads übernommen und daraus eine neue Alignment-Zeile erzeugt. Diese wird an das Ende der SAM-Datei geschrieben und die Variablen, welche die Felder repräsentieren, werden neu initialisiert. Die Initialisierung der Alignment-Zeilen enthält unter anderem auch das Setzen der Read-spezifischen Daten. Das sind alle Daten aus der Datenbank, die innerhalb desselben Reads identisch sind. Liegt kein Wechsel der Read-ID vor, so ist die Alignment-Zeile noch nicht abgeschlossen und alle Felder, die nicht Read-spezifisch sind, müssen auf Basis der Daten aus der aktuellen Iteration angepasst werden. Das Konzept dieser Anpassungen wurde in Kapitel 4.1.3 erläutert und wird hier direkt in Programmcode umgesetzt. Sobald die Schleife terminiert, ist auch die Berechnung der SAM-Datei abgeschlossen.

4.2.6 IGV Controller

Der IGV Controller wird durch den Parameter -i aktiviert, wenn der *SimpleSamExportParser* verwendet wird. Wenn aktiviert, wird das Untermodul aufgerufen, nachdem die SAM-Datei erfolgreich exportiert wurde. Die Kommunikation erfolgt über die Port-Schnittstelle von IGV [40]. Auf der Internetseite vom IGV[6] befindet sich ein Überblick über die verfügbaren Befehle, die über die Port-Schnittstelle gesendet werden können.

Voraussetzungen Für das erfolgreiche Ansteuern des IGV ist Voraussetzung, dass dieser gestartet und die Port-Schnittstelle in den Einstellungen aktiviert ist. Der konfigurierte Port im IGV muss dabei mit der Einstellung der Variable igv_port des Variablen-Manager von CoGaDB übereinstimmen. Standardmäßig wird der Port 60151 zur Kommunikation

[5] http://www.omg.org/spec/UML/
[6] http://www.broadinstitute.org/software/igv/PortCommands

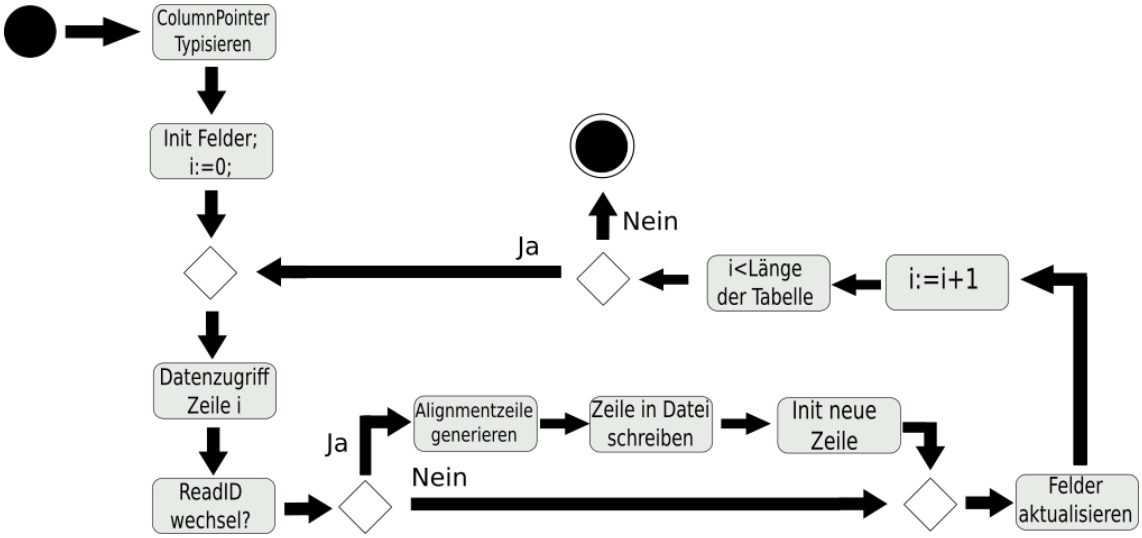

Abbildung 4.5: Berechnung der Alignment-Zeilen

verwendet. Das zur exportierten SAM-Datei passende Reference-Genom muss außerdem im Vorfeld im IGV importiert worden sein. Weil der IGV-Befehl zum Auswählen des Reference-Genoms den Wert von rg_name verwendet, muss sichergestellt sein, dass der *Unique Identifier* beim Importieren identisch zum rg_name in CoGaDB ist. Eine genaue Anleitung zur Konfiguration der IGV-Schnittstelle befindet sich auf der Wiki-Seite des CoGaDB-SAM-Moduls[7].

Ablauf Der IGV Controller sortiert und indiziert zunächst die exportierte SAM-Datei mithilfe des Zusatzprogramms IGVtools, falls diese entsprechend der Konfigurationsanleitung platziert wurde. Diese Indizierung wird von IGV verlangt und erhöht unter anderem die Performance[8]. Wird igvtools nicht gefunden, so kann die Datei wegen fehlender Sortierung und Indizierung nicht geladen werden.
Anschließend wird die Port-Schnittstelle des IGV benutzt, um die Befehle an den IGV zu übertragen. Dafür werden die Befehle direkt auf den Socket geschrieben [40] Der IGV Controller überträgt auf diese Weise nacheinander die folgenden Befehle:

1. new
2. genome <referenceGenomeName>
3. load <samFilePath>
4. goto <locus>
5. snapshotDirectory <snapshotDirectory>

[7]https://bitbucket.org/bress/cogadb/wiki/CoGaDB-SAM-Modul
[8]vgl. https://www.broadinstitute.org/software/igv/BAM

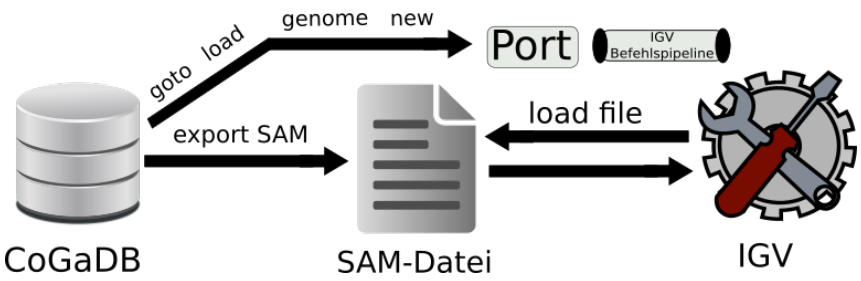

Abbildung 4.6: Überblick IGV-Controller

6. snapshot <snapshotName>

IGV bestätigt jeden eingegangenen Befehl mit einer Bestätigung und hängt diesen an das Ende der internen Befehlspipeline. Durch Abwarten der Bestätigung wird sichergestellt, dass die Pipeline nicht überläuft und keine Befehle verloren gehen. IGV handelt diese Befehle nacheinander aus der Befehlspipeline ab. Zunächst wird durch den Befehl *new* eine neue Sitzung gestartet. Durch den Befehl *genome <referenceGenomeName>* lädt IGV das passende Reference-Genom. Dieses muss wie zuvor beschrieben im IGV importiert sein. Den Namen des Referenzgenoms erhält man mithilfe der SQL Query aus Listing 4.3.

```
1 select rg_name
2 from reference_genome;
```

Listing 4.3: IGV Query

Daher ist es wichtig, dass der angegebene Name beim Datenbank-Import mit dem im IGV übereinstimmt. Der Befehl *load <samFilePath>* lädt die exportierte SAM-Datei als Sample-Genom in den IGV. Als <samFilePath> wird dabei der Pfad zu der exportierten SAM-Datei benutzt. Dieser kann aus dem entsprechenden Attribut des Command Parser abgefragt werden. Durch *goto <locus>* zeigt IGV das Genom an der angegebenen Stelle an. Der Locus hat die Form <contigID:minPos-maxPos> und wird so gesetzt, dass das erste Contig der SAM-Datei angezeigt wird, das Genomdaten enthält. In diesem Contig wird der Bereich ausgewählt, den der Benutzer angegeben hat. Auch diese Daten können aus dem Command Parser abgefragt werden. Sollte der Benutzer den Bereich mit dem Spezialsymbol '*' angegeben haben, so wird das gesamte Contig angezeigt.
Wenn der Parameter -s ebenfalls aktiviert wurde, so wird zuletzt mittels der Befehle *snapshotDirectory <snapshotDirectory>* und *snapshot <snapshotName>* noch ein Bildschirmfoto des derzeit angezeigten Teil des Genoms gemacht. Die Werte für snapshotDirectory und snapshotName werden dabei aus dem Variablen-Manager von CoGaDB übernommen.

4.3 Zusammenfassung und Ausblick

Dieses Kapitel hat einen Integrationsansatz vorgestellt, mit dessen Hilfe bestehende Werkzeuge der Genomforschung so an ein DBMS angebunden werden können, dass dieses zum Datenmanagement verwendet wird. Dafür wurde eine DBMS-SAM-Schnittstelle vorgestellt. Diese wurde zunächst konzeptionell entworfen und anschließend an einem konkreten Beispiel umgesetzt.
Eine Erweiterung bzw. Verbesserung der Implementierung ist durch diverse Ansätze denkbar. Weitere Argument Parser können hinzugefügt werden, um auch eine komplexere Auswahl der Daten zu ermöglichen. Durch eine Parallelisierung des Algorithmus zur Berechnung der Alignment-Zeilen kann eventuell eine höhere Performance auf Multi-Core-Rechnern erzielt werden. Die Erweiterung der DBMS-SAM-Schnittstelle um weitere Parameter kann zusätzliche Funktionen freischalten, darunter auch die Anbindung an weitere Werkzeuge der Genomforschung.
Es muss noch sichergestellt werden, dass das beschriebene Konzept die Integration korrekt und effektiv umsetzt. Daher wird es im folgenden Kapitel anhand festgelegter Kriterien evaluiert und bewertet.

Kapitel 5

Bewertung des SAM-Exports

Um den praktischen Nutzen des vorgestellten Integrationsansatzes zu untersuchen, wird dieser im Folgenden anhand der drei Kriterien Korrektheit, Laufzeit und Skalierbarkeit analysiert. Die Korrektheit der Algorithmen wird anhand von Testverfahren untersucht. Die Laufzeitanalyse basiert auf konkreten Messungen, die im Rahmen eines Experiments durchgeführt und ausgewertet werden. Das Kapitel schließt mit der Untersuchung der Skalierbarkeit, die durch eine strukturelle Analyse des Konzepts und der Implementierung durchgeführt wird. Die Bewertung wird hauptsächlich auf der konkreten Implementation aus Kapitel 4 durchgeführt.

5.1 Korrektheit

Es muss sichergestellt werden, dass die exportierte SAM-Datei konsistent ist und die Genomdaten korrekt exportiert werden. Die Konsistenz der SAM-Datei kann anhand der SAM-Format-Spezifikation [39] durchgeführt werden. Alternativ existieren auch Kommandozeilentools wie *Picard*[1], die automatisierte Routinen besitzen, um die Konsistenz von SAM-Dateien zu untersuchen. Um zu überprüfen, ob die Genomdaten korrekt und vollständig exportiert werden, muss die SAM-Datei zeilenweise mit den Daten in der Datenbank verglichen werden. Diese Überprüfung kann durch den Einsatz von automatisierten Testverfahren vereinfacht werden.

5.1.1 Header-Bereich

Der Header ist zwar ein optionaler Zusatz des SAM-Formats, allerdings werden in der SAM-Format-Spezifikation einige Zusammenhänge zwischen den Header-Feldern definiert. Kapitel 2.2 gibt einen umfassenden Überblick über das SAM-Format. Es ist zum Beispiel erforderlich, dass jede @SQ Zeile die Tags *SN* und *LN* enthält [39]. Bei der Konsistenz-Überprüfung des Headers muss überprüft werden, dass solche formalen Zusammenhänge

[1] https://broadinstitute.github.io/picard/index.html

eingehalten werden. Die in Kapitel 4.2 vorgestellte Implementation schreibt zuerst immer die Zeile *@HD VN:1.5 SO:coordinate*. Der Versionstag gibt an, in welcher Version der SAM-Spezifikation die Datei geschrieben wurde. Weil die aktuelle Version 1.5 ist, ist dieser Tag offensichtlich korrekt. Die Sortierreihenfolge SO wird auf *coordinate* gestellt. Dadurch wird ausgedrückt, dass die exportierte SAM-Datei in der Reihenfolge der zugehörigen @SQ-Zeilen nach RNAME sortiert ist [39]. Die *Order-By-Klauseln* in beiden Queries sortieren die Daten primär nach dem Attribut c_id. Dadurch wird sichergestellt, dass diese Sortierung im Header-Bereich derjenigen im Alignment-Bereich gleicht und somit die Sortierreihenfolge *coordinate* eingehalten wird. Die *@SQ-Zeilen* müssen, wenn sie angegeben werden, als minimale Informationen die Tags *SN* und *LN* besitzen. In *SN* wird der Name eines Contigs gespeichert und *LN* enthält dessen Basenanzahl im Reference-Genom. Dabei erhält jedes Contig eine eigene *SQ-Zeile* [39]. Die Header Query liefert die benötigten Informationen direkt. Das Attribut c_name aus dem verwendeten Genomdatenschema entspricht dem *SN-Tag* des SAM-Formats. Um den Wert des *LN-Tags* zu erhalten, muss für jedes Contig die Anzahl der Basen gezählt werden. Dafür wird die Anzahl der Einträge des Attributs *rb_ id* mit der SQL-Aggregatfunktion *count* gezählt. Dadurch, dass die Header Query nach c_id gruppiert, wird sichergestellt, dass für jedes Contig genau eine @SQ-Zeile ausgegeben wird. Also ist sichergestellt, dass der Header-Bereich der exportierten SAM-Datei konsistent ist und der Format-Spezifikation entspricht.
Es bleibt zu zeigen, dass die Daten auch korrekt exportiert wurden und den Daten im DBMS gleichen. Weil die Daten jedoch ohne weitere Konversion direkt aus der Ergebnistabelle der Header Query übernommen werden und diese nach oben genannter Argumentation der SAM-Format-Spezifikation entsprechen, ist die Korrektheit des Headers gegeben, wenn das DBMS keine Fehler in der Bearbeitung der Header Query macht. Dies kann jedoch für die Evaluation des Integrationsansatzes vorausgesetzt werden, weil der Fehler dann in der Implementation des DBMS liegen würde.

5.1.2 Alignment-Bereich

Der Alignment-Bereich besitzt, wie in Kapitel 2.2 beschrieben, elf Pflichtfelder. In einigen dieser Felder kann jedoch ein Default-Wert gesetzt werden, falls die zu dem Feld gehörige Information nicht vorhanden ist. Um die Korrektheit des Alignment-Bereichs zu untersuchen, wird zwischen zwei Szenarien unterschieden. Das erste Szenario besteht darin, die Korrektheit zu überprüfen, nachdem die komplette Datenbank exportiert wurde. Im zweiten Szenario wird nur ein Teil der Datenbank exportiert.

Voller Export Wenn das gesamte Genom aus der Datenbank exportiert wird, kann die exportierte SAM-Datei mit der ursprünglichen SAM-Datei verglichen werden. Dabei werden die Genomdaten zunächst aus der ursprünglichen SAM-Datei in die Datenbank importiert und anschließend über den Export-Algorithmus exportiert. Sind diese beiden

Dateien funktional identisch, d. h. die relevanten Felder der SAM-Datei gleich, so war der Export für diesen Datensatz korrekt. Nach der Mengentheorie sind zwei Mengen identisch, wenn jede Menge Teilmenge der anderen ist. Also muss sichergestellt werden, dass jede Zeile der ursprünglichen SAM-Datei eine entsprechende funktional identische Zeile in der exportierten SAM-Datei enthält und umgekehrt. Damit dieser Ansatz funktioniert, wird vorausgesetzt, dass der Import-Algorithmus korrekt arbeitet.
Je nach Datenbankschema werden nicht alle Daten des SAM-Formats in der Datenbank gespeichert. Die Genomerweiterung von CoGaDB beschränkt sich zum Zeitpunkt dieser Arbeit beispielsweise nur auf gemappte Reads. Ungemappte Reads werden nicht unterstützt. Auch kann eine SAM-Datei Kommentare für einzelne Werkzeuge der Genomforschung enthalten. Auch für diese ist in dem eingesetzten Genomdatenmodell kein Attribut vorgesehen. Der Vergleich der SAM-Datei darf daher nur die Felder der SAM-Datei vergleichen, die durch das verwendete Datenmodell unterstützt werden und für die weitere Verarbeitung der exportierten Datei relevant sind. Diese Werte von Hand zu vergleichen ist aufgrund der Größe der beiden Dateien sehr mühsam, weshalb es sich anbietet, eine automatisierte Routine zu entwickeln, die zwei SAM-Dateien auf Basis der relevanten Felder miteinander vergleicht. Im Kapitel 5.1.3 wird daher eine automatisierte Methode vorgestellt, die zwei SAM-Dateien auf Basis der von der CoGaDB-Genomerweiterung unterstützten Felder miteinander vergleicht. Felder, bei denen vom CoGaDB-SAM-Modul ein Default-Wert gesetzt wird, werden nicht verglichen.

Teil Export Die Verifikation eines Teilexportes kann auch über den Vergleich mit der Ursprungsdatei realisiert werden. Es muss jedoch zusätzlich beachtet werden, dass durch die Angabe einer minimalen und maximalen Position auch Teilstücke eines Reads exportiert werden können. Für den Vergleich eines Teilexports mit der ursprünglichen vollständigen SAM-Datei muss demnach überprüft werden, ob die exportierte SAM-Datei eine Art Teilmenge der vollständigen SAM-Datei ist. Dafür muss jede Alignment-Zeile der exportierten SAM-Datei eine entsprechende Alignment-Zeile in der vollständigen SAM-Datei haben. Anders als beim vollen Export müssen die Zeilen jedoch nicht funktional identisch sein. Es genügt, wenn eine Art Teilmengenbeziehung besteht. Die Zeile des Teilexports enthält alle Daten der Zeile, die der vollständigen SAM-Datei angehört. Auch muss jede Alignment-Zeile der vollständigen SAM-Datei eine entsprechende Zeile in der exportierten SAM-Datei haben. Abschließend muss noch sichergestellt werden, dass die exportierten Daten ausschließlich aus dem angegebenen Bereich sind und, dass auch alle Daten dieses Bereichs exportiert werden. Eine automatisierte Routine, die einen solchen Vergleich übernimmt, ist um ein vielfaches komplexer als bei einem vollen Export, weshalb diese im Rahmen dieser Arbeit nicht implementiert wird.
Eine wesentlich einfachere Möglichkeit besteht darin, zunächst aus der Ursprungsdatei die Daten aus dem angegebenen Bereich 'herauszuschneiden'. Für diese Aufgabe existieren

bereits bestehende Werkzeuge. Eines davon ist das in Kapitel 2.2 beschriebene Werkzeug *SAMtools* [20]. Anschließend kann die oben beschriebene Methode des vollen Exports verwendet werden, um den Vergleich durchzuführen. Dazu wird das von *SAMtools* herausgeschnittene Teilstück als Referenzdatei genommen. Dieser Ansatz setzt voraus, dass das eingesetzte Werkzeug zum Herausschneiden korrekt arbeitet. Weil jedoch Werkzeuge wie *SAMtools* sehr verbreitet sind, um SAM- oder BAM-Dateien zu manipulieren, kann man von einer sehr geringen Fehlerwahrscheinlichkeit ausgehen.

5.1.3 SAM-Verifikator

Der *SAM-Verifikator* ist eine automatisierte Routine, die entwickelt wurde, um zwei SAM-Dateien auf funktionelle Gleichheit zu untersuchen. Dabei werden die Eigenschaften der CoGaDB-Genomerweiterung berücksichtigt und Felder, die mit Default-Werten gesetzt werden, nicht verglichen. Auch die Header-Zeilen werden nicht verglichen, da diese ein optionaler Zusatz des SAM-Formats sind. Der *SAM-Verifikator* ist ein eigenes Modul, das für CoGaDB exemplarisch implementiert wurde, um die Korrektheit der exportierten SAM-Dateien durch den gesamten Entwicklungsprozess des CoGaDB-SAM-Moduls sicherzustellen. Der *SAM-Verifikator* kann mit dem Befehl *sam_verificator <originalSam> <exportedSam>* direkt aus der CoGaDB-Commandline aufgerufen werden. Ein detaillierteres Beispiel befindet sich auf der Wiki-Seite des CoGaDB-SAM-Moduls[2].

Funktionsweise Der SAM-Verifikator bekommt den Dateipfad von zwei SAM-Dateien übergeben. Der erste Dateipfad zeigt auf die originale SAM-Datei und der zweite Dateipfad auf die SAM-Datei, die zuvor von dem CoGaDB-SAM-Modul exportiert wurde. Die Original-SAM-Datei wird zunächst zeilenweise in eine Datenstruktur im Hauptspeicher geladen. Ungemappte Zeilen, das heißt Zeilen, die im Feld *Flag* das '0x4'-Bit gesetzt haben [39] oder im Feld CIGAR den Default-Wert '*' haben, werden vom CoGaDB SAM/-BAM Importer nicht importiert. Diese werden daher beim Laden herausgefiltert und nicht in die Datenstruktur geladen. Auch die Elemente des erweiterten Cigar-Strings werden derzeit nicht von der CoGaDB Genomerweiterung unterstützt. Daher werden Basen, die mit 'S', 'H', 'P' oder 'N' markiert sind, nicht importiert. Damit der Vergleich erfolgreich ist, müssen die Zeilen beim Laden so modifiziert werden, dass aus dem erweiterten Cigar ein Standard Cigar wird. Dafür müssen Basen, die mit *Soft Clipping (S)* markiert sind, aus den Feldern SEQ, QUAL und CIGAR herausgeschnitten werden. Basen, die mit *Hard Clipping (H)*, *Padding (P)* oder *Skipped Region (N)* markiert sind, müssen lediglich aus dem Feld CIGAR herausgeschnitten werden, da diese in den anderen Feldern nicht vorkommen. Detaillierte Informationen zum Cigar-String bzw. SAM-Format befinden sich in Kapitel 2.2.

[2] https://bitbucket.org/bress/cogadb/wiki/CoGaDB-SAM-Modul

Nachdem alle Zeilen der Original-SAM in der Datenstruktur sind, wird je eine Zeile der exportierten SAM-Datei eingelesen und eine entsprechende Zeile in der Datenstruktur gesucht. Dabei werden die einzelnen Felder der Zeilen miteinander verglichen. Alle Felder bis auf RNEXT, PNEXT und TLEN müssen identisch sein, damit zwei Zeilen zusammenpassen. Die drei Ausnahmen kommen daher, dass verbundene Reads derzeit noch nicht vollständig von der CoGaDB-Genomerweiterung unterstützt werden. Sobald eine passende Zeile in der Datenstruktur gefunden wurde, wird diese entfernt und die nächste Zeile der exportierten SAM-Datei eingelesen. Wird für eine Zeile aus der exportierten SAM-Datei keine entsprechende Zeile in der Datenstruktur gefunden, ist der Export nicht korrekt. Wenn die Datenstruktur nicht leer ist, nachdem die exportierte SAM vollständig abgearbeitet wurde, so ist der Export nicht vollständig.
Wenn zu jeder Zeile aus der exportierten SAM-Datei eine entsprechende Zeile in der Datenstruktur gefunden wird und sich in dieser keine Elemente mehr befinden, sind die beiden SAM-Dateien funktional identisch und der Export war korrekt und vollständig.

Ausgabe Der *SAM-Verifikator* gibt alle Zeilen der exportierten SAM-Datei aus, zu denen keine entsprechende Zeile in der Datenstruktur gefunden werden konnte. Am Ende gibt der *SAM-Verifikator* auch noch alle Zeilen der Datenstruktur aus, die übrig bleiben. Außerdem gibt der *SAM-Verifikator* aus, ob die beiden SAM-Dateien bis auf nicht-unterstützte Elemente identisch sind oder nicht, ob der Export also erfolgreich war.

Analyse und Ausblick Der SAM-Verifikator dient ausschließlich dazu, die Korrektheit des CoGaDB-SAM-Moduls anhand automatisierter Tests zu analysieren. Die Worst-Case-Komplexität liegt im quadratischen Bereich. Diese wird allerdings nur angenommen, wenn die beiden überprüften SAM-Dateien grundlegend verschieden sind, also nie eine passende Zeile vom Algorithmus gefunden wird. Wenn die beiden SAM-Dateien identisch und in gleicher Reihenfolge sortiert sind, so liegt eine lineare Komplexität vor. Ein größeres Problem ist der hohe Ressourcenverbrauch. Die Original-SAM-Datei wird unkomprimiert in den Hauptspeicher geladen, was bei großen Dateien Schwierigkeiten bereiten wird, vor allem, wenn zusätzlich eine Datenbank geladen ist.
Die Komplexität kann verbessert werden, wenn der SAM-Verifikator nur anzeigen soll, ob zwei Dateien identisch sind, und nicht, welche Zeilen problematisch sind. Dann könnte der Algorithmus terminieren, sobald die erste Zeile gefunden wurde, zu der keine passende existiert. Wenn die beiden SAM-Dateien zuvor auch noch in der gleichen Reihenfolge sortiert werden, ergibt sich durch die Sortierung eine Worst-Case-Komplexität von $\mathcal{O}(n \log n)$. Um den Ressourcenverbrauch im Hauptspeicher zu reduzieren, kann der Algorithmus auf der Festplatte arbeiten und immer nur eine Zeile jeder Datei gleichzeitig in den Hauptspeicher laden. Weil die Datenmengen, die im Kontext dieser Arbeit verglichen werden mussten, relativ klein waren, wurde auf diese Verbesserungen der Einfachheit halber verzichtet.

5.1.4 Grenzen der Testverfahren

Die automatisierten Testverfahren dienen zur Qualitätssicherung des entwickelten CoGaDB-SAM-Moduls. Durch Testen kann niemals bewiesen werden, dass ein Testobjekt keine Fehlerzustände besitzt. Es wird lediglich die Wahrscheinlichkeit gesteigert, diese zu finden [38, S. 37]. Daher handelt es sich bei der hier eingesetzten Korrektheitsanalyse keinesfalls um einen Beweis. Dennoch kann man davon ausgehen, dass durch die durchgeführten Tests die Wahrscheinlichkeit für Fehlerwirkungen deutlich verringert werden konnte. Durch die Automatisierung der Tests mithilfe des SAM-Verifikator-Moduls kann zusätzlich die Zuverlässigkeit der Tests verbessert werden, besonders, weil große Datenmengen verglichen werden [38, S. 209]. Außerdem helfen automatisierte Tests zu einer verbesserten Wartbarkeit, weil diese nach Änderungen des Systems ohne großen Aufwand durchgeführt werden können und sicherstellen, dass die Änderung zumindest an den getesteten Stellen keine neuen Fehlerzustände erzeugt.

5.2 Laufzeitanalyse

Wie in Kapitel 3.3 beschrieben, wird für den Integrationsansatz durch Exportfunktion in ein universelles Dateiformat der Daten-Transfer-Overhead am problematischsten bewertet. Aus diesem Grund wird dieser hier noch einmal im Detail an einer konkreten Implementierung evaluiert und mit einer bestehenden Lösung verglichen.

5.2.1 Pretests

Während der Implementation des CoGaDB-SAM-Moduls wurden Pretests durchgeführt. Dadurch konnte untersucht werden, ob Implementierungsdetails negative Auswirkungen auf die Laufzeit haben. Diese Stellen konnten im Laufe des Implementationsprozesses noch verbessert werden. Dafür wurden Laufzeitmessungen und Profiling-Logs ausgewertet. Diese Pretest laufen auf dem vollständigen *Harrington Genom* und es werden immer gesamte Contigs exportiert. Die Contigs wurden so gewählt, dass unterschiedliche Anzahl von Basen, sowie eine unterschiedliche Position im Genom vorliegt. Die Position im Genom kann über die ContigID ermittelt werden. Je Größer die ContigID, desto weiter hinten im Genom befindet sich dieser Ausschnitt.

Ergebnisse

Tabelle 5.1 zeigt die Ergebnisse des ersten Pretests. Es sind zwei Auffälligkeiten zu erkennen. Zum einen besteht eine starke Korrelation zwischen der Position im Genom und der Laufzeit der SAM-Datei-Berechnung und zum anderen haben die SQL-Queries eine überdurchschnittlich lange Laufzeit.

ContigID	Anzahl Basen	Query-Laufzeit	SAM-Datei-Berechnung
6	9392	147,50s	0,21s
29	27719	150,39s	0,56s
100	19617	146,37s	0,93s
47648	3262	147,51s	60,84s
85378	1398	149,40s	49,72s

Tabelle 5.1: Ergebnisse Pretest vor Optimierung

1. Position im Genome Contigs, die weiter hinten im Genom liegen, benötigen deutlich Länger in der Berechnung der SAM-Datei. Das Contig mit der ID 47648 benötigt zum Beispiel fast 300 mal so lange, wie der Contig mit der ID 6. Und das obwohl letzteres beinahe dreimal so viele Basen exportiert.
Der Grund für diesen Zusammenhang liegt in der Kompression der Genomdaten und den Zugriffszeiten auf die komprimierten Datensätze. Für die CoGaDB Genomerweiterung werden die Daten für jede Spalte individuell komprimiert. Weil CoGaDB Daten spaltenbasiert speichert, können Kompressionsverfahren eingesetzt werden, die sich nicht auf die Query-Laufzeit auswirken [10]. Daher ist die Laufzeit für die Dekompression der Daten nicht in der Laufzeit der Query, sondern in der Laufzeit der SAM-Datei-Berechnung zu finden. Dort werden auf die komprimierten Tabellenfelder zugegriffen und diese in diesem Kontext dekomprimiert. Als Kompressionsmethode wird hauptsächlich *run-length-encoding* eingesetzt. Dabei werden wiederholt vorkommende Werte ersetzt, indem der Wert 'w' und die Anzahl der Wiederholungen 'c' angegeben wird. Um jetzt auf den Wert einer Zeile mit ID n zuzugreifen, müssen so lange die 'c'-Werte aufsummiert werden, bis n erreicht wurde. Je höher die ZeilenID ist, desto länger dauert das berechnen der Summe.

2. Query Laufzeit Die Laufzeit der Queries ist durch das gesamte Testset zwar einigermaßen konstant, jedoch mit ca. 150 Sekunden sehr hoch. Der Hauptgrund für die hohe Laufzeit ist, dass innerhalb der Alignment-Query vier Tabellen gejoined werden. Weil Genomdaten meist sehr groß sind, enthalten vor allem die Tabellen *sample_ base* und *reference_ base* mehrere Milliarden Einträge. Daher ist die Laufzeit dieser joins sehr hoch.

Verbesserungen

Das CoGaDB-Team hat basierend auf den oben genannten Ergebnissen die Implementation für das Szenario des SAM-Exports optimiert. Das Kompressionsverfahren wurde auf eine sortierbare Methode des *run-length-encoding* umgestellt. Dabei wird nicht mehr die Anzahl der Wiederholungen eines Wertes gespeichert. Stattdessen wird die erste ID, die nicht mehr Teil dieses Wiederholungszyklus ist, gespeichert. Dadurch kann eine binäre Suche eingesetzt werden, um den richtigen Wert zu finden, was deutlich effizienter ist. Für

ContigID	Anzahl Basen	Query-Laufzeit	SAM-Datei-Berechnung
6	9392	23,22s	0,001s
29	27719	27,83s	0,01s
100	19617	22,80s	0,01s
47648	3262	26,81s	0,08s
85378	1398	27,57s	0,02s

Tabelle 5.2: Ergebnisse Pretest nach Optimierung

alle weiteren Zeilen kann ausgenutzt werden, dass der Zugriff sequenziell erfolgt, also immer aufeinanderfolgende Zeilen angefragt werden. Daher kann bei der vorher berechneten Summe begonnen werden. Durch ein einfaches Caching kann sichergestellt werden, dass auch wenn der Zugriff nur beinahe sequentiell ist, die Summe nicht jedes mal von vorne berechnet werden muss. Sobald jedoch eine Zeile angefragt wird, die nicht Nachfolger einer gecachten Zeile ist, muss die binäre Suche von vorne beginnen. In Tabelle 5.2 sieht man, dass sich durch dieses Verfahren die Laufzeit der SAM-Datei-Berechnung für Contigs, die weiter hinten im Genom liegen, signifikant verbessert hat.
Das CoGaDB-Team hat außerdem die Laufzeit der Queries durch interne Optimierungsverfahren deutlich reduziert. Dabei wurde vor allem die Laufzeit der Joins verbessert.

5.2.2 Versuchsaufbau und Messverfahren

In der Genomanalyse sind häufig nur kleine Bereiche des Genoms relevant. Bei vielen Algorithmen und Analysen ergibt es wenig Sinn, diese auf dem gesamten Genom laufen zu lassen, wenn die Betrachtung eines kleinen Bereichs ausreichend wäre. Ein Beispiel dafür ist die Visualisierung der Genomdaten mit IGV. Aus diesem Grund wird dieser Versuch so aufgebaut, dass die Laufzeit gemessen wird, welche benötigt wird, einen Bereich des Genoms mit Hilfe der in Kapitel 4.2 vorgestellte Implementierung des CoGaDB-SAM-Moduls zu exportieren. Vergleichend wird die Laufzeit von SAMtools evaluiert, um den selben Bereich aus der BAM-Datei des gesamten Genoms herauszuschneiden. Wie in Abbildung 5.1 zu erkennen, ist das Laden der SAM-Datei in den IGV unabhängig von der Methode, wie diese zuvor generiert wurde. Deswegen ist diese Laufzeit für den Versuchsaufbau nicht signifikant.

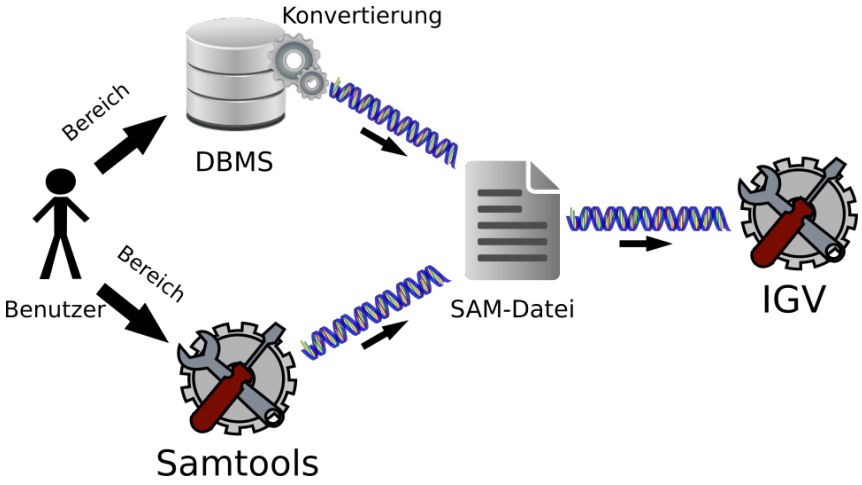

Abbildung 5.1: Evaluation des CoGaDB-SAM-Moduls gegen SAMtools

Mit SAMtools kann nur ausgewählt werden, ob der Header des gesamten Genoms oder gar kein Header in die SAM-Datei geschrieben werden soll. Es gibt keine Funktion, die nur den Header des angefragten Bereichs exportiert. Wenn der Header des gesamten Genoms in die SAM-Datei geschrieben wird, entsteht ein unnötiger Overhead, sowohl in der Laufzeit, als auch in der Speichergröße. Weil außerdem die Visualisierung der Genome mit IGV auch ohne den Header funktioniert, wird dieser im Rahmen des Versuchs nicht berücksichtigt. Dazu wird SAMtools so konfiguriert, dass kein Header in die SAM-Datei geschrieben werden soll. Für das CoGaDB-SAM-Modul wird die Laufzeit der Header-Query aus der Gesamtlaufzeit herausgerechnet. Der Schreibvorgang des Headers ist für kleine Exports zu vernachlässigen.

Preprocessing

SAMtools kann einen Bereich nur aus einer SAM- oder BAM-Datei herausschneiden, wenn diese Datei sortiert ist und einen Index hat[3]. Diese Sortierung und Indizierung muss für jedes Genom einmalig durchgeführt werden und fällt daher unter Preprocessing.
In CoGaDB existiert ein vergleichbares Preprocessing. Hier muss jedes Genom einmalig importiert werden. Dazu werden die Genomdaten für das relationale Genomdatenmodell aufbereitet und in dieses eingelesen.
Im Rahmen dieses Versuchs wird zusätzlich auch das Preprocessing beider Ansätze miteinander verglichen.

Testumgebung

Die Laufzeitmessungen werden auf einer Serverarchitektur mit 48 CPU-Kernen und 256GB Hauptspeicher durchgeführt. Das Betriebssystem ist Ubuntu in der Version 14.04. Als

[3]vgl. SAMtools manual: http://www.htslib.org/doc/samtools.html

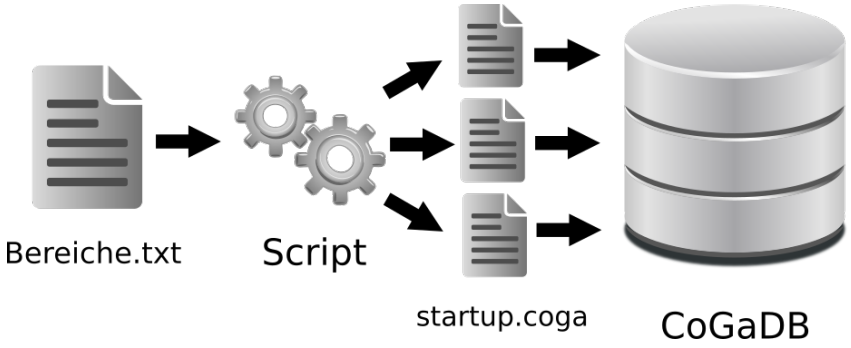

Abbildung 5.2: Aufbau Script zur Laufzeitmessung

Software wird CoGaDB in der Version 0.4.1 ohne GPU-Beschleunigung verwendet. Weil das selbst entwickelte CoGaDB-SAM-Modul benötigt wird, muss CoGaDB vom Quellcode compiliert werden. Als Compiler-Parameter wurde *'-DCMake_BUILD_TYPE=Release -DENABLE_GPU_ACCELERATION=OFF'* verwendet. Für die Laufzeitmessung wurde eigens der CoGaDB-Branch *sam_exporter_experiments* erstellt, um den Softwarestand der Messungen festzuhalten. Es besteht außerdem eine Abhängigkeit zu dem Hype-Branch *Development*. Weitere Informationen zur Konfiguration und Nutzung von CoGaDB können der Bedienungsanleitung [4] bzw. dem User-Guide [5] entnommen werden.
Um die Laufzeitmessungen in CoGaDB durchzuführen, wurde ein Skript entwickelt. Abbildung 5.2 zeigt einen Überblick über dieses. Das Skript liest die zu untersuchenden Bereiche aus einer Datei ein. Für jeden dieser Bereiche wird eine startup.coga generiert und CoGaDB mit dieser gestartet. Das Ergebnis wird auf den Standard-Output-Stream und kann von da in eine Datei umgeleitet werden. Jeder Bereich wird dabei zehnmal gemessen.

Beschreibung der Genome

Die Versuche werden auf dem Genom eines Gerstenkorns der Sorte *Harrington* durchgeführt. Das ist eine Getreidesorte, die zwei Reihen Körner hat und im Frühling blüht. Diese Gerstensorte wurde im März 1981 von der Universität in Sashatchewan entdeckt [16]. Das Genom umfasst ca. 5 Billionen Basenpaare [13]. Das Sample Genome für die Versuche wurde vom IPK Gatersleben[6] bereitgestellt.
Um sicherzustellen, dass die Laufzeiten nicht von der Wahl des Genoms abhängen, werden außerdem Versuche auf dem menschlichen Genom durchgeführt. Dazu werden die Genomdaten verwendet, die durch das 1000-Genom-Projekt zur Verfügung gestellt wurden. Das Ziel dieses Projektes war, das Genom von 1000 verschiedenen Menschen aus unterschiedlichen Bereichen der Welt mit Hilfe von Next-Generation-Sequencing zu sequenzieren [41].

[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/gpu/cogadb/0.3/refman.pdf
[5] http://cogadb.cs.tu-dortmund.de/wordpress/wp-content/uploads/2015/03/cogadb_user_guide.pdf
[6] http://www.ipk-gatersleben.de/

Anzahl exportierter Basen pro Contig	vorderes Drittel	mittleres Drittel	hinteres Drittel
ca. 100	contig_14	contig_1052435	contig_2670662
	contig_22	contig_1052592	contig_2670520
ca. 1000	contig_236	contig_1052720	contig_2667824
	contig_695	contig_1052921	contig_2668467
ca. 10.000	contig_288	contig_1052878	contig_2667401
	contig_687	contig_1058079	contig_2666401
ca. 100.000	contig_1286	contig_1059281	contig_2667744
	contig_5076	contig_1063772	contig_2661389

Tabelle 5.3: Phase 1: Ausgewählte Contigs des Harrington Genoms

Als Sample-Genome wurde das Genome mit der Kennung *HG00096* verwendet[7]. Das verwendete Referenzgenome wurde ebenfalls vom 1000-Genome-Projekt bereitgestellt[8].

Stichprobe

Es werden die folgenden vier Thesen aufgestellt, anhand derer die Auswahl der Stichprobe abgeleitet werden soll:

1. These 1: Die Laufzeit ist abhängig von der Position im Genom.
2. These 2: Die Laufzeit ist abhängig von der Anzahl der exportierten Basen.
3. These 3: Die Laufzeit ist abhängig von der Anzahl der exportierten Contigs.
4. These 4: Die Laufzeit ist abhängig von der Größe des Genoms

Um diese Thesen zu beantworten, gliedert sich das Experiment in drei Phasen.

Phase 1 Die ersten beiden Thesen können in derselben Phase überprüft werden. Dazu wird das Harrington-Genom zunächst in die drei Bereiche 'vorderes Drittel', 'mittleres Drittel' und 'hinteres Drittel' aufgeteilt, indem die Anzahl der Contigs des Genoms durch drei dividiert wird. Anschließend wird aus jedem der drei Bereiche Contigs ausgewählt, wo die Anzahl der exportierten Basen annähernd 100, 1000, 10.000 und 100.000 ist. Dabei wird darauf geachtet, dass die ausgewählten Contigs innerhalb eines Bereichs relativ nahe beieinander liegen, um eine bessere Vergleichbarkeit zu garantieren. Tabelle 5.3 zeigt die für diese Phase ausgewählten Contigs des Harrington-Genoms.

[7] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/data/HG00096/alignment/HG00096.mapped.ILLUMINA.bwa.GBR.lo
[8] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/human_g1k_v37.fasta.gz

	Block A	**Block B**	**Block C**	**Block D**
Contigs	contig_184 contig_390 contig_435 contig_1286 contig_4258	contig_4871 contig_5076 contig_5602 contig_5626 contig_5637	contig_5661 contig_5662 contig_5669 contig_5671 contig_5718	contig_5672 contig_5674 contig_5679 contig_5686 contig_5692
Anzahl exportierter Basen	1.009.504	941.151	1.110.961	1.090.802

Tabelle 5.4: Phase 2: Ausgewählte Contigs von Block A - D

Phase 2 In der zweiten Phase wird die dritte These untersucht. Dazu werden je fünf ausgewählte Contigs zu einem Block zusammengeschlossen. Diese Contigs sind so gewählt, dass die Anzahl der exportierten Basen eines Blocks ca. 1 Million ergibt. Außerdem wird darauf geachtet, dass der Abstand im Genom zwischen den Contigs nicht zu groß ist, damit das Ergebnis dadurch nicht beeinträchtigt wird. In dem Versuch wird jeder der Blocks zunächst einzeln und anschließend mehrere Blocks zusammengefasst gemessen. Auf diese Weise kann der Overhead untersucht werden, der entsteht, wenn mehrere Contigs gleichzeitig exportiert werden. Tabelle 5.4 zeigt, welche Contigs für diese Phase ausgewählt wurden.

Phase 3 Die dritte Phase dient vor allem dazu, die vierte These zu untersuchen. Allerdings soll auch sichergestellt werden, dass Ergebnisse nicht von dem Harrington Genom abhängen. Daher sollen auch weitere Ergebnisse für die ersten beiden Thesen gesammelt werden, auf Basis eines zweiten Genoms. Weil die Contigs des menschlichen Genoms sehr groß sind, ist ein Vergleich mit den Laufzeiten des Harrington Genoms nicht durch den Export von vollen Contigs möglich. Daher müssen die Contigs durch die Angabe einer minimalen bzw. maximalen Position so beschränkt werden, dass die gewünschte Anzahl von Basen exportiert wird. Es werden die Contigs mit den Kennungen 0, 22 und 70 gewählt. Dabei wird zunächst ein kompletter Export gemessen. Anschließend wird auch die Laufzeit von Teilstücken dieser Contigs untersucht.

Messverfahren

Die Laufzeit wird auf Verhältnisskalenniveau gemessen. Dies setzt voraus, dass zwischen Merkmalsausprägungen nicht nur eine Ordnung besteht, sondern auch die Größe des Unterschieds messbar ist. Außerdem muss ein inhaltlich bedeutungsvoller Nullpunkt existieren [36]. Diese Voraussetzung ist bei der Laufzeit offensichtlich gegeben. 0 Sekunden Laufzeit werden dabei als Nullpunkt angesetzt. Durch das Messen auf Verhältnisskalenniveau kön-

nen die verschiedenen Laufzeiten verhältnismäßig miteinander verglichen werden [36]. Jede Messung wird zehnmal wiederholt und anschließend der Mittelwert als Ergebnis genommen. Wenn innerhalb der Wiederholungen starke Ausreißer vorkommen, werden die Messungen für dieses Testdatum wiederholt. Die Reliabilität soll außerdem über die Retest-Methode ermittelt werden. Dabei wird die Messung mit den gleichen Testdaten nach einem gewissen Zeitintervall wiederholt und die Ergebnisse miteinander verglichen. Dadurch kann sichergestellt werden, dass die gemessenen Ergebnisse stabil und unabhängige von äußeren Faktoren der Testdurchführung sind.

5.2.3 Ergebnisse und Bewertung

Preprocessing Abbildung 5.3 zeigt den Vergleich zwischen dem Preprocessing von CoGaDB und von SAMtools. Es lässt sich erkennen, dass die Laufzeit des Preprocessing für das menschliche Genom sowohl für SAMtools, als auch für CoGaDB um den Faktor 2,85 größer als beim Preprocessing für das Harrington-Genom. Dies liegt vor allem daran, dass das menschliche Sample-Genom 3,4 mal mehr Basen hat. Der Unterschied im Verhältnis liegt daran, dass das Harrington-Genom aus deutlich mehr Contigs besteht und dadurch ein zusätzlicher Overhead entsteht.

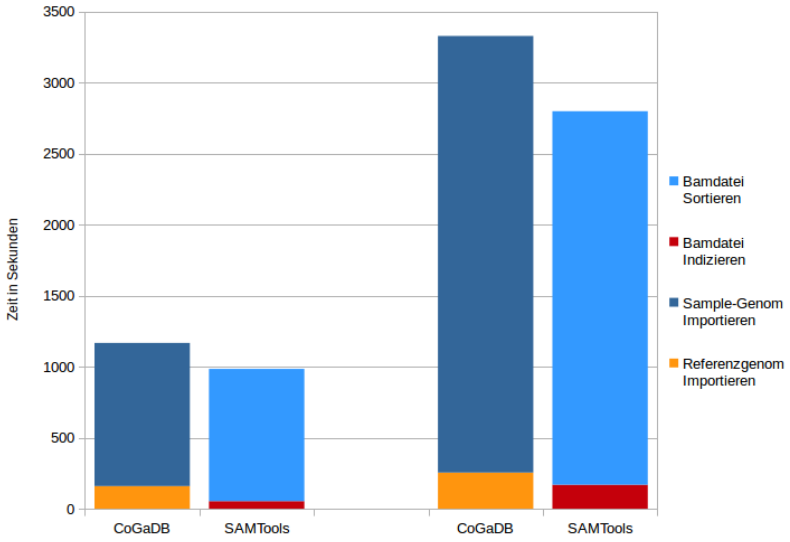

Abbildung 5.3: Vergleich des Preprozessing anhand des Human- und Harrington-Genoms

Für CoGaDB wird die Laufzeit des Preoprocessing durch das Importieren des Sample-Genoms dominiert. Der Grund dafür ist, dass das Datenvolumen des Sample-Genoms durch die sich überlappenden Reads deutlich größer ist als das des Referenzgenoms. Auch müssen mehr Informationen für das Sample-Genom als für das Referenzgenom gespeichert werden

(vgl. Kap. 2.4). Bei SAMtools wird das Preprocessing durch das Sortieren der BAM-Datei dominiert. Das Indizieren schlägt nach der Sortierung kaum noch ins Gewicht.
Insgesamt ist das Preprocessing für CoGaDB um den Faktor 1,18 langsamer als für SAMtools. Dieses Verhältnis bleibt auch bei beide Genomen, die jeweils unterschiedlich groß sind, stabil.

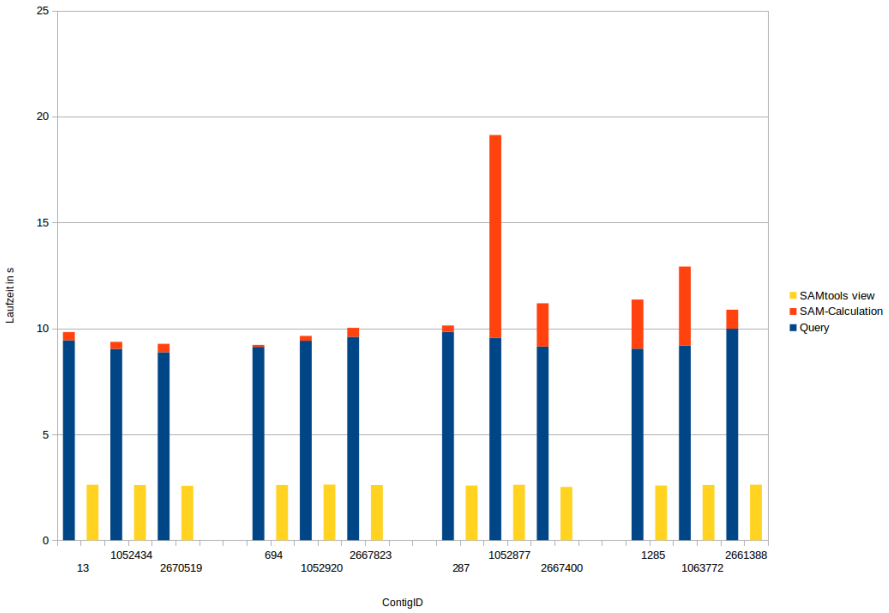

Abbildung 5.4: Abhängigkeit von der Anzahl der Contigs gruppiert nach der Laufzeit

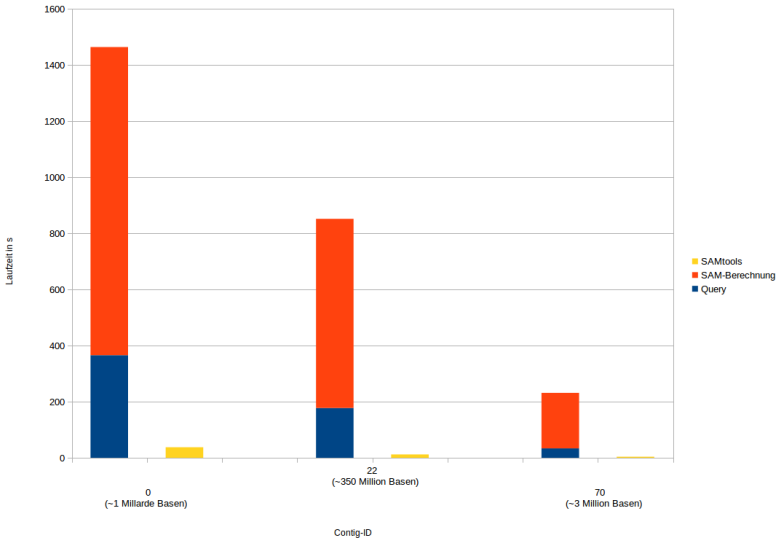

Abbildung 5.5: Vergleich des Exports ganzer Contigs vom Human-Genom

Allgemeine Beobachtungen Abbildung 5.4 zeigt die Laufzeitmessungen des Harrington-Genoms gruppiert nach Anzahl der exportierten Basen. Für die Auswertung wurde jeweils der Wert mit der höheren Laufzeit gewählt, damit für das Testset eine Worst-Case Laufzeit angegeben werden kann. Die Laufzeit von SAMtools ist über alle Bereiche relativ stabil. Für CoGaDB lassen sich deutliche Schwankungen erkennen, die individuell vom einzelnen Contig abhängen. Die Query läuft einigermaßen stabil, ist jedoch mit knapp unter zehn Sekunden relativ langsam. In den meisten Fällen wird die Laufzeit bei kleinen Exports von der Query dominiert. Die große Laufzeit kommt daher zustanden, dass für die Joins in der Alignment-Query die Datensätze dekomprimiert werden müssen, wodurch ein zusätzlicher Aufwand an Rechenleistung entsteht. Außerdem ist das *joinen* von großen Tabellen generell sehr teuer. Für die SAM-Berechnung entsteht die Laufzeit vor allem durch den Zugriff auf die komprimierten Spalten. Dabei kann dieser, wenn die Basen alle nahe beieinander liegen, durch das nach den Pretests eingeführte Caching sehr effizient sein. In einigen Fällen ist der Zugriff jedoch nicht sequenziell, sodass die Laufzeit der SAM-Berechnung signifikant ansteigt. Insgesamt ist die Laufzeit von CoGaDB für die gemessenen Exporte mit bis zu 100.000 Basen im Worst-Case um den Faktor 7 langsamer. Im Durchschnitt liegt dieser Faktor jedoch zwischen drei und vier.
In Abbildung 5.5 ist der Vergleich zwischen CoGaDB und SAMtools für ganze Contigs des menschlichen Genoms zu sehen. Die Contigs sind insgesamt größer als die des Harrington-Genoms. Das Contig mit der ID 0 besitzt ca. 1 Milliarde Basen, das Contig mit der ID 22 besitzt ca. 250 Millionen Basen und das Contig mit der ID 70 ca. 3 Millionen Basen. Es ist zu erkennen, dass bei größeren Exports die Laufzeit durch die SAM-Berechnung dominiert wird. Für die gemessenen großen Exporte mit bis zu 1 Milliarden Basen ist der Export mit CoGaDB um den Faktor 40-70 größer als bei SAMtools.

These 1 Die erste These ist, dass die Laufzeit von der Position im Genom abhängt. Jede Gruppe der Abbildung 5.4 zeigt die Laufzeit für ein Contig am Anfang, in der Mitte und am Ende des Genoms, bei jeweils gleicher Anzahl exportierter Basen. In diesem Diagramm ist kein direkter Zusammenhang zwischen der Position im Genom und der Laufzeit zu erkennen. Zwar sind einige Contigs deutlich länger als andere, jedoch lässt sich kein klarer Zusammenhang erkennen. Allerdings ist die Laufzeit der SAM-Berechnung für diese Anzahl der Basen auch geringer als die Messschwankungen. Aus diesem Grund sieht man in Abbildung 5.6 noch einmal die Laufzeit für größere Exporte des menschlichen Genoms gruppiert nach Position im Genom. Diese Exporte haben eine Größe von 1, 2 und 3 Millionen Basen. In dieser Messung lässt sich ein Zusammenhang zwischen der Position im Genom und der Laufzeit der SAM-Berechnung feststellen. Dieser kommt daher, dass das Dekomprimieren für Spalten, die weiter hinten im physischen Speicher liegen mehr Rechenzeit benötigen als für Spalten, die vorne liegen und die Position im Genom direkt mit der Position im physischen Speicher zusammenhängt.

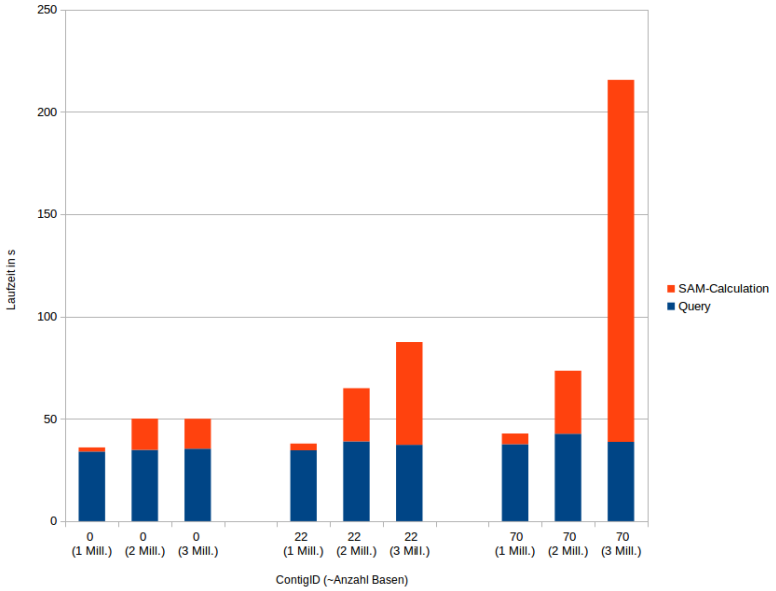

Abbildung 5.6: Vergleich Laufzeit Human-Genom verschiedene Größen und Positionen

These 2 Die zweite These sagt aus, das ein Zusammenhang zwischen der Laufzeit und der Anzahl der exportierten Basen besteht. In Abbildung 5.4 und 5.6 ist die Laufzeit der Query relativ stabil. Allerdings ist in diesen Abbildungen der Unterschied in der Anzahl der Basen sehr gering. In Abbildung 5.5 ist der Unterschied größer. Hier lässt sich auch erkennen, dass die Query-Laufzeit mit der Anzahl der exportierten Basen wächst. Wieder liegt der Grund vor allem an den 'Joins' in der Alignment-Query. Wenn größere Datenbereiche angefragt werden, sind auch die Tabellen größer, die gejoined werden. Dadurch erhöht sich auch die Rechenzeit. Ähnliches gilt für die Laufzeit der SAM-Berechnung. Bei größerer Datenmenge entsteht deutlich mehr Aufwand beim Dekomprimieren der Spalten. Außerdem erhöht sich auch der Aufwand beim Zusammensetzen der SAM-Zeilen und dem Schreiben in die Datei.

These 3 Diese These ist, dass die Laufzeit von der Anzahl der exportierten Contigs abhängt. Abbildung 5.7 zeigt die Laufzeiten der SAM-Berechnung der in Kapitel 5.2.2 definierten Blöcke, gruppiert nach der Anzahl der Blöcke. In jeder Gruppe sieht man auf der linken Seite zunächst die Laufzeiten, wenn die einzelnen Blöcke einzeln exportiert und anschließend die Laufzeiten addiert werden. Auf der rechten Seite sieht man die Laufzeit, wenn alle Blöcke exportiert werden. Es ist kein zusätzlicher Overhead in der SAM-Berechnung zu erkennen, wenn mehrere Contigs gleichzeitig exportiert werden.

These 4 Die letzte These sagt aus, dass die Laufzeit von der Größe des Genoms abhängig ist. Abbildung 5.8 zeigt den Export von 100.000 Basen am Anfang, in der Mitte und

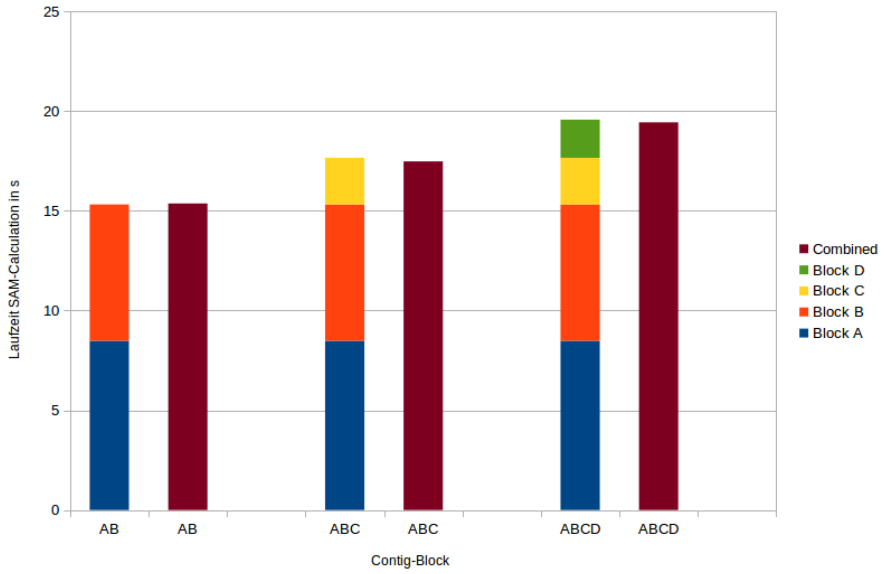

Abbildung 5.7: Vergleich der SAM-Berechnung mit unterschiedlicher Anzahl der Contigs

am Ende des Genoms. Für das menschliche Genom wurde passgerechte Stücke aus den jeweiligen Contigs herausgeschnitten. Es ist eine deutliche Abhängigkeit der Query-Laufzeit von der Größe des Genoms zu erkennen. Diese begründet sich vor allem dadurch, dass die Genome und Contigs des menschlichen Genoms deutlich größer sind als die des Harrington-Genoms und dadurch die Joins in der Alignment-Query teurer werden.

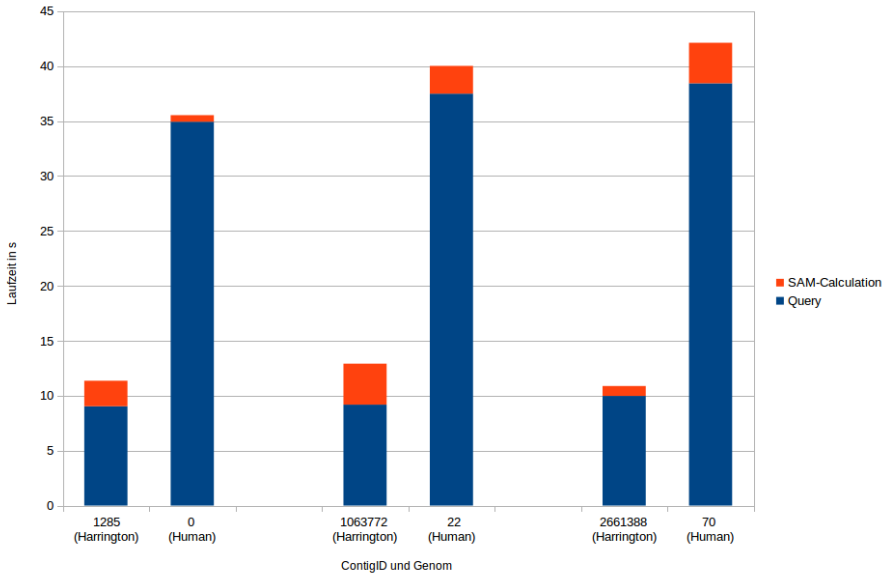

Abbildung 5.8: Vergleich zwischen Harrington-Genom und Human-Genom mit 100.000 Basen

Zusammenfassung Zusammenfassend lässt sich sagen, dass das Preprozessing für CoGaDB ein wenig mehr Rechenzeit benötigt, als das Preprozessing von SAMtools. Weil der Unterschied jedoch nicht so groß ist und außerdem das Preprozessing für jedes Genom nur einmal durchgeführt werden muss, ist der Unterschied zu vernachlässigen.
Für kleinere Bereiche von bis zu 100.000 Basen entsteht eine akzeptable Laufzeit beim Export aus CoGaDB. CoGaDB ist zwischen dreimal und siebenmal langsamer als SAMtools, was durch eine effizientere Implementierung kompensiert werden kann. Die Laufzeit wird vor allem durch die Query dominiert. Wenn große Bereiche exportiert werden, ist die Laufzeit von CoGaDB deutlich ineffizienter als die von SAMtools. Diese wird hauptsächlich durch den Zugriff auf die komprimierten Spalten der Ergebnistabelle und die in diesem Kontext stattfindende Dekompression der Daten dominiert.
Die Laufzeit der Query ist abhängig von der Anzahl der exportierten Basen, wächst jedoch nur sehr langsam. Bei Größenunterschiede von bis zu einer Millionen Basen scheint die Laufzeit stabil. Die SAM-Berechnung ist abhängig von der Position im Genom und von der Anzahl der exportierten Basen, nicht aber von der Anzahl der exportierten Contigs.

5.3 Skalierbarkeit

Die Skalierbarkeit eines Systems definiert sich durch seine Anpassung an eine wachsende Anzahl von Elementen sowie den Nutzen durch den Einsatz von besserer Hardware [1]. Bondi unterteilt die Skalierbarkeit dabei in die vier Bereiche *Load Scalability*, *Space Scalability*, *Space-time Scalability* und *Structural Scalability* [1]. Die Eingabegröße des SAM-Export-Algorithmus setzt sich aus der Größe des zu exportierenden Bereichs sowie der Größe des Genoms in der Datenbank zusammen.

5.3.1 Load Scalability

Ein Programm wird laut Bondi als *Load Scalable* bezeichnet, wenn dieses mit kleinen, mittleren und großen Input-Größen gleichermaßen und ohne zusätzlichen Ressourcenaufwand zurechtkommt. Kriterien sind dabei z. B. das Ausnutzen von Parallelität und das effiziente Scheduling für den Zugriff auf gemeinsam genutzte Ressourcen. [1] Die vorgestellte konzeptionelle Umsetzung des SAM-Exports auf RDBMS ist durchaus in der Lage, eine gute *Load Scalability* vorzuweisen. Die Datenextraktion beruht auf Queries an ein DBMS. Deren Implementation ist in der Regel sehr gut skalierbar und kann auch mit großen Datenmengen problemlos umgehen. In der Datenkonversation wird jeder Read individuell aus den Tabellenzeilen mit der jeweiligen Read-ID berechnet. Da keine Abhängigkeit besteht, kann dies parallelisiert durchgeführt werden. Auch die Berechnung der Header-Zeilen kann parallel zu der Berechnung der Alignment-Zeilen durchgeführt werden, da beide auf dem Ergebnis von verschiedenen SQL Queries arbeiten. Es existieren zwei Herausforderungen, die zu lösen sind, bevor von einer guten *Load Scalability* gesprochen werden kann.

Es muss zum einen dafür gesorgt werden, dass die Reads nach der parallelen Berechnung am Ende sortiert in eine Datei geschrieben werden. Zum anderen muss die Abhängigkeit der Laufzeit der Datenextraktion von der Größe der Datenbank eingedämmt werden. Die für diese Arbeit exemplarische Umsetzung am Beispiel von CoGaDB macht jedoch keinen Nutzen von Multi-Core-Rechnern, da im Quellcode zur Datenkonversion keine Parallelmechanismen eingebaut sind. Auch wird die Query-Berechnung und SAM Kalkulation bei wachsender Datenbankgröße deutlich ineffizienter, obwohl die Größe der extrahierten Daten gleichbleibend ist. Diese Punkte sorgen dafür, dass die Load Scalability durch eine andere Implementation noch verbessert werden könnte.

5.3.2 Space Scalability

Als *Space Scalable* bezeichnet Bondi ein Programm, bei dem der genutzte Speicher nicht mehr als sublinear zu der Eingabegröße wächst. [1] Der SAM-Export kann als Space Scalable bezeichnet werden. Die Speichergröße der fertigen SAM-Datei ist linear abhängig von der Größe des zu exportierenden Bereichs. Außerdem ist der Speicherbedarf der SAM-Datei unabhängig von der Größe der Datenbank. Weil außerdem jede Zeile direkt nach der Berechnung in die Datei geschrieben wird, ist der benötigte Hauptspeicher sogar relativ unabhängig von der Eingabegröße. Bei wachsender Größe des zu exportierenden Bereichs steigt lediglich die Größe der Zeigerstruktur auf die Daten. Wenn die Vorschläge zur Verbesserung der *Load Scalability* umgesetzt werden, wird die *Space Scalability* negativ beeinflusst. Ein solcher statistisch negativer Zusammenhang zwischen *Space Scalability* und *Load Scalability* ist in der Praxis keine Ausnahme. [1]

5.3.3 Space-time Scalability

Die *Space-time Scalability* eines Programms bemisst sich in der Fähigkeit, auch große Eingabemengen effizient zu bearbeiten. [1] In der oben vorgestellten Implementation des SAM-Exports am Beispiel von CoGaDB ist der Umgang mit sehr großen Datenmengen problematisch. Die Laufzeit der Datenextraktion, d. h. der SQL Queries, verschlechtert sich für eine große Datenbankgröße signifikant. Die Laufzeit der Datenkonversation für den Full Export eines ganzen Genoms ist nicht im Bereich des praktisch Nutzbaren. Die Vermutung legt jedoch nahe, dass durch eine effizientere Implementation deutlich bessere Ergebnisse erreicht werden können. Das Ergebnis der SQL Query kann durch einmaliges Durchlaufen in einer Schleife als SAM-Datei exportiert werden. Daher befindet sich die Komplexität im linearen Bereich, wodurch die Space-time Scalability akzeptabel ist. Eine parallele Durchführung der Datenkonversation kann auch hier die Space-time Scalability verbessern.

5.3.4 Structural Scalability

Structural Scalability untersucht, ob die Größe der zu verarbeitenden Objekte durch die Implementation oder die eingesetzten Standards beschränkt ist citeBondi00. Weil jeder Eintrag einer Tabelle im DBMS eine individuelle ID bekommt, besteht eine gewisse Beschränkung durch die Größe des Datentyps, der zum Verwalten der IDs eingesetzt wird. Weil dies jedoch interne Mechanismen des DBMS sind, die nicht von der Implementation des SAM-Moduls abhängen, wird dieser Punkt nicht näher betrachtet.

5.3.5 Zusammenfassung

Insgesamt wird die Skalierbarkeit deutlich von Implementierungsdetails beeinflusst. Eine mehr auf Skalierbarkeit bedachte Umsetzung der in Kapitel 4.1 beschriebenen Konzepte könnte den Umgang mit wachsenden Datenmengen oder die Effizienz auf wachsender Hardware deutlich verbessern. Ein zentrales Konzept zur Verbesserung der Load bzw. Spacetime Scalability ist dabei das Ausnutzen von Multicore-Prozessoren durch den Einsatz von Parallelberechnungen. Allerdings muss darauf geachtet werden, dass dabei nicht zu viel an Space Scalability verloren geht. Der genötigte Speicherbedarf auf der Festplatte oder im Hauptspeicher ist in der vorgestellten Implementierung sehr gering.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse dieser Arbeit zusammen. Zunächst werden die definierten Ziele und Forschungsfragen wiederholt. Anschließend werden diese Fragen anhand der Zusammenfassung beantwortet. Das Kapitel schließt mit einem Ausblick.

Ziele der Arbeit Um den Einsatz von DBMS in die Praxis zu etablieren, ist es erforderlich, dass Genomforscher in der Übergangsphase ihre gewohnten Werkzeuge und Analyseketten einsetzen können. Dafür muss gewährleistet sein, dass alle wichtigen Werkzeuge an ein DBMS angebunden werden können. Um dies zu überprüfen wurden folgende Forschungsfragen aufgestellt:

1. *Wie lassen sich möglichst alle bestehenden Werkzeuge am vielversprechendsten an ein DBMS anbinden?*
2. *Wie lässt sich eine solche Anbindung implementieren?*
3. *Wie lässt sich eine solche Anbindung evaluieren?*
4. *Wie wird die Laufzeit einer solchen Anbindung im Vergleich zum aktuellen Stand der Technik bewertet?*
5. *Woraus resultiert diese Laufzeit?*

Zusammenfassung Um diese Fragen zu beantworten wurden zunächst in Kapitel 2 die Grundlagen für die weitere Arbeit geschaffen. Darunter fielen zum Beispiel die Grundbegriffe der Genomforschung, das SAM-Format und die Speicherung von Genomdaten in relationalen Datenbankmanagementsystemen. In Kapitel 3 wurden verschiedene Ansätze zur Anbindung bestehender Werkzeuge diskutiert und anhand einer Bewertungsskala ein Ansatz ausgewählt. Es wurde entschieden, dass der Export in ein universelles Datenformat am vielversprechendsten ist (Forschungsfrage 1). In Kapitel 4 wurde ein Konzept aufgestellt, wie sich diese Anbindung umsetzen lässt. Außerdem wird eine konkrete Implementierung am Beispiel von CoGaDB und IGV angegeben (Forschungsfrage 2). Diese Umsetzung

wurde anschließend in Kapitel 5 evaluiert. Dabei wurde diskutiert, wie die Korrektheit sichergestellt und die Skalierbarkeit bewertet werden kann. Die Korrektheit kann mittels des SAM-Verifikator in automatisierte Softwaretests eingebaut werden. Die Skalierbarkeit ist noch nicht optimal. Gerade bei großen Datenmengen kann es zu Problemen kommen (Forschungsfrage 3). Außerdem wird die Laufzeit des CoGaDB-SAM-Moduls im Vergleich zu SAMtools bewertet. Dabei kam heraus, dass für kleinere Exporte mit bis zu 100.000 Basen der Export um den Faktor vier bis sieben langsamer ist als bei SAMtools. Diese Laufzeit wird durch die Query dominiert. Beim Export von größeren Bereichen ist die Laufzeit signifikant schlechter und wird von der SAM-Berechnung dominiert (Forschungsfrage 4). Bei der Query resultiert die Laufzeit vor allem durch die 'Joins' in der Alignment-Query. Joins von großen Datenmengen sind ohnehin schon teuer. Außerdem werden die Daten dabei dekomprimiert. Es besteht ein Zusammenhang zwischen der Anzahl der exportierten Basen und der Laufzeit der Query. Bei der SAM-Berechnung wird die Laufzeit hauptsächlich von dem Zugriff auf die komprimierten Zeilen dominiert. Dadurch ist die Laufzeit abhängig von der Position im Genom und der Anzahl der exportierten Basen (Forschungsfrage 5).

Ausblick Wenn große Bereiche exportiert werden, ist die Laufzeit für einen Einsatz in der Praxis noch zu lang. Daher muss diese noch verbessert werden. In der derzeitigen Implementierung müssen die Spalten vor einem 'Join' zunächst dekomprimiert werden. Wenn der 'Join' so umgesetzt wird, dass er auch ohne Dekompression funktioniert, kann erwartet werden, dass die Laufzeit der Query noch einmal signifikant verbessert wird. Außerdem kann die Laufzeit und Skalierbarkeit durch eine parallele Umsetzung des SAM-Export-Moduls noch weiter verbessert werden. Im Laufe dieser Arbeit wurde der IGV exemplarisch angebunden. Es können weitere Werkzeuge angebunden werden und diese einzeln evaluiert werden.

Anhang A

Weitere Informationen

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[2] E. Brent and P. Green. Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome Res8*, pages 186–194, 2014.

[3] S. Breß. The design and implementation of cogadb: A column-oriented gpu-accelerated DBMS. *Datenbank-Spektrum*, 14(3):199–209, 2014.

[4] Y. Bromberg. Building a genome analysis pipeline to predict disease risk and prevent disease. *Journal of Molecular Biology*, page 3993–4005, 2013.

[5] R. Brouver, M. van den Hout, F. Grosveld, and W. IJcken. Narwhal, a primary analysis pipeline for ngs data. *Bioinformatics vol. 28*, pages 284–285, 2011.

[6] T. Carver, U. Böhme, T. D. Otto, J. Parkhill, and M. Berriman. Bamview: viewing mapped read alignment data in the context of the reference sequence. *Bioinformatics vol. 26*, pages 676–677, 2010.

[7] G. Chiang, P. Clapham, et al. Implementing a genomic data management system using irods in the wellcome trust sanger institute. *BMC Bioinformatik*, 12, 2011.

[8] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res*, 38(6):1767–1771, April 2010.

[9] S. Dorok. Towards genome analysis on modern database systems. Master thesis, University of Magdeburg, Germany, November 2013.

[10] S. Dorok. The relational way to dam the flood of genome data. In *Proceedings of the 2015 ACM SIGMOD on PhD Symposium*, SIGMOD '15 PhD Symposium, pages 9–13, New York, NY, USA, 2015. ACM.

[11] S. Dorok, S. Breß, H. Läpple, and G. Saake. Toward efficient and reliable genome analysis using main memory database systems. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 34:1–34:4. ACM, 2014.

[12] S. Dorok, S. Breß, and G. Saake. Toward efficient variant calling inside main-memory database systems. In *International Workshop on Biological Knowledge Discovery and Data Mining (BIOKDD-DEXA)*, pages 41–45. IEEE, 2014.

[13] S. Dorok, S. Breß, J. Teubner, and G. Saake. Flexible analysis of plant genomes in a database management system. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 509–512, 2015.

[14] P. Flicek. Sense from sequence reads: methods for alignment and assembly. *Nature Methods*, 6:6–12, 2009.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Prentice Hall, New Jersey, 1 edition, 1994.

[16] B. L. HARVEY and B. G. ROSSNAGEL. Harrington barley. *Canadian Journal of Plant Science*, 64(1):193–194, 1984.

[17] D. W. Hoffmann. *Software-Qualität.* Springer, Berlin Heidelberg, 2 edition, 2013.

[18] ISO/IEC 9126-1. Software engineering - product quality international standard, 2001.

[19] T. Komatsuda, M. Pourkheirandish, C. He, et al. Six-rowed barley originated from a mutation in a homeodomain-leucine zipper i-class homeobox gene. *PNAS*, pages 1424–1429, 2007.

[20] H. Li, B. Handsaker, A. Wysoker, et al. The sequence alignment/map format and samtools. *Bioinformatics vol. 25*, pages 2078–2079, 2009.

[21] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefing in Bioinformatics*, 11:473–483, 2010.

[22] H. Li, J. Ruan, and R. Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18:1851–1858, 2008.

[23] E. R. Mardis. The $1,000 genome, the $100,000 analysis? *Genome Medicine*, 2, 2010.

[24] R. Merkl. *Bioinformatik: Grundlagen, Algorithmen, Anwendungen.* Wiley-VCH, Weinheim, 3 edition, 2015.

[25] M. L. Metzker. Sequencing technologies – the next generation. *Nature Reviews Genetics*, 11:31–46, 2010.

[26] I. Milne, M. Bayer, L. Cardle, et al. Tablet- next generation sequence assembly visualization. *Bioinformatics vol. 26*, pages 401–402, 2010.

[27] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis.* CSHL Press, New Yourk, 2 edition, 2004.

[28] J. W. Nicol, G. A. Helt, S. G. Blanchard, A. Raja, and A. E. Loraine. The integrated genome browser: free software for distribution and exploration of genome-scale datasets. *Bioinformatics vol. 25*, pages 2730–2731, 2009.

[29] R. Nielsen, J. Paul, A. Albrechtsen, and Y. Song. Genotype and snp calling from next-generation sequencing data. *Nature Reviews. Genetics*, 12(6):443–451, 2011.

[30] J. O'Rawe, T. Jiang, G. Sun, et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome Medicine*, 5:28.

[31] S. Pabinger, A. Dander, et al. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefing in Bioinformatics*, 15:256–278, 2013.

[32] J. Qi, F. Zhao, A. Buboltz, and S. C. Schuster. ingap: an integrated next-generation genome analysis pipeline. *Bioinformatics vol. 26*, pages 127–129, 2010.

[33] A. Rheinländer, M. Knobloch, N. Hochmuth, and U. Leser. Prefix tree indexing for similarity search and similarity joins on genomic data. *SSDBM*, pages 519–536, 2010.

[34] J. T. Robinson, H. Thorvaldsdottir, W. Winckler, et al. Integrative genomics viewer. *Nature Biotechnology 29*, pages 24–26, 2011.

[35] U. Röhm and J. Blakeley. Data management for high-throughput genomics. *CIDR*, 2009.

[36] P. Sedlmeier and F. Renkewitz. *Forschungsmethoden und Statistik.* Pearson, München, 2 edition, 2013.

[37] S. Shah, Y. Huang, T. Xu, et al. Atlas – a data warehouse for integrative bioinformatics. *BMC Bioinformatics.*

[38] A. Spillner and T. Linz. *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB-Standard.* dpunkt, Heidelberg, 5 edition, 2012.

[39] The SAM/BAM Format Specification Working Group. Sequence alignment/map format specification, August 2015.

[40] H. Thorvaldsdottir, J. T. Robinson, and J. P. Mesirov. Integrative genomics viewer (igv): high-performance genomics data visualization and exploration. *Briefings in Bioinformatics vol. 14*, pages 178–192, 2012.

[41] M. Via, C. Gignoux, and E. G. G. Burchard. 1000 genome project: New opportunities for research and social challanges. *Genome Medicine*, page 2(1):3, 2010.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 7. Oktober 2015

John Sarrazin