

University of London  
Imperial College of Science, Technology and Medicine  
Department of Computing

# **Big tranSMART for Clinical Decision Making**

Shicai Wang

Submitted in part fulfilment of the requirements for the degree of  
Doctor of Philosophy in Computing of the University of London and  
the Diploma of Imperial College, September 2015



## Abstract

Molecular profiling data based patient stratification plays a key role in clinical decision making, such as identification of disease subgroups and prediction of treatment responses of individual subjects. Many existing knowledge management systems like tranSMART enable scientists to do such analysis. But in the big data era, molecular profiling data size increases sharply due to new biological techniques, such as next generation sequencing. None of the existing storage systems work well while considering the three "V" features of big data (Volume, Variety, and Velocity).

New Key Value data stores like Apache HBase and Google Bigtable can provide high speed queries by the Key. These databases can be modeled as Distributed Ordered Table (DOT), which horizontally partitions a table into regions and distributes regions to region servers by the Key. However, none of existing data models work well for DOT.

A Collaborative Genomic Data Model (CGDM) has been designed to solve all these issues. CGDM creates three Collaborative Global Clustering Index Tables to improve the data query velocity. Microarray implementation of CGDM on HBase performed up to 246, 7 and 20 times faster than the relational data model on HBase, MySQL Cluster and MongoDB. Single nucleotide polymorphism implementation of CGDM on HBase outperformed the relational model on HBase and MySQL Cluster by up to 351 and 9 times. Raw sequence implementation of CGDM on HBase gains up to 440-fold and 22-fold speedup, compared to the sequence alignment map format implemented in HBase and a binary alignment map server. The integration into tranSMART shows up to 7-fold speedup in the data export function. In addition, a popular hierarchical clustering algorithm in tranSMART has been used as an application to indicate how CGDM can influence the velocity of the algorithm. The optimized method using CGDM performs more than 7 times faster than the same method using the relational model implemented in MySQL Cluster.



## **Copyright Declaration**

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.



## Acknowledgements

I would like to express my thanks to:

- My supervisor, Professor Yike Guo, a talented teacher and passionate scientist. Without his continuous support and encouragement, this thesis would not be possible.
- All my friends and colleagues in Imperial College London, Akara Supratak, Axel Oehmichen, Bertan Kavuncu, Chao Wu, Chun-Hsiang Lee, David Birch, David Johnson, Diana O'Malley, Dilshan Silva, Florian Guitton, Ibrahim Emam, Ioannis Pandis, Kai Sun, Lei Nie, Li Guo, Manohara Rukshan Batuwita, May Yong, Michelle Osmond, Mihaela Mares, Moustafa Ghanem, Orestis Tsinalis, Peter Rice, Rui Han, Scott Chih-Hsi Kuo, Shulin Yan, Sijin He, Simiao Yu, Xian Yang, Xi-angchuan Tian, Xiaoping Fan, Yajie Ma and Yang Li.
- My dearest father, though he cannot see my achievements any more, every achievement I made depends on what he has taught me.
- My dearest mother, for her constant support and encouragement.





## **Dedication**

*To my dearest Baba and Mama.*

*'Let there be light: and there was light.'*

*the Bible*

# Contents

<b>Abstract</b>	<b>i</b>
<b>Copyright Declaration</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Objectives . . . . .	1
1.1.1 Microarray . . . . .	2
1.1.2 SNP . . . . .	3
1.1.3 Raw Sequence . . . . .	4
1.1.4 Application . . . . .	5
1.2 Contributions . . . . .	6
1.3 Organization . . . . .	7
1.4 Statement of Originality . . . . .	8
1.5 Publications . . . . .	9

---

<b>2</b>	<b>Background</b>	<b>11</b>
2.1	tranSMART architecture . . . . .	11
2.1.1	Overview . . . . .	11
2.1.2	Data model . . . . .	13
2.1.3	Future tranSMART . . . . .	18
2.2	Big Data Management . . . . .	21
2.2.1	HBase: An open source implementation of Google Bigtable . . . . .	22
2.2.2	CCIndex . . . . .	25
2.2.3	MongoDB . . . . .	28
<b>3</b>	<b>Generic -omics data model</b>	<b>30</b>
3.1	Introduction . . . . .	30
3.2	Methods . . . . .	33
3.2.1	Fault Tolerance . . . . .	35
3.3	Generic data model for molecular profiling data . . . . .	36
3.3.1	Subject table . . . . .	36
3.3.2	Position table . . . . .	39
3.3.3	Cross-study index table . . . . .	40
3.4	Performance evaluation methods . . . . .	41
3.5	Conclusion . . . . .	42

---

<b>4</b>	<b>Microarray Key Value Model</b>	<b>43</b>
4.1	Introduction . . . . .	43
4.1.1	Classic relational data model - I2B2 . . . . .	45
4.1.2	TranSMART high dimensional data model . . . . .	48
4.2	Key Value Microarray Model . . . . .	52
4.2.1	Subject table . . . . .	52
4.2.2	Position Table . . . . .	57
4.2.3	Cross-study Table . . . . .	58
4.3	Performance Evaluation . . . . .	59
4.3.1	Experimental Environment . . . . .	59
4.3.2	Query by subject . . . . .	60
4.3.3	Query by probe within a study . . . . .	66
4.3.4	Query by probe across studies . . . . .	67
4.4	Implementation on tranSMART . . . . .	68
4.4.1	Implementation . . . . .	68
4.4.2	Performance evaluation . . . . .	70
4.5	Discussion . . . . .	75
4.6	Conclusion . . . . .	77
<b>5</b>	<b>SNP Key Value Model</b>	<b>78</b>
5.1	Introduction . . . . .	78
5.1.1	dbSNP . . . . .	81

---

5.1.2	VCF format . . . . .	83
5.1.3	HapMap . . . . .	85
5.1.4	TranSMART SNP model . . . . .	86
5.1.5	SeqWare . . . . .	88
5.2	SNP Key Value Model . . . . .	89
5.2.1	Subject Table . . . . .	90
5.2.2	Position Table . . . . .	91
5.2.3	Cross-study Table . . . . .	93
5.3	Performance Evaluation . . . . .	95
5.3.1	Query by subject . . . . .	96
5.3.2	Query by SNP position . . . . .	99
5.4	Integration into tranSMART . . . . .	101
5.4.1	Benchmark . . . . .	103
5.4.2	Database configuration . . . . .	104
5.4.3	Results . . . . .	104
5.5	Discussion . . . . .	104
5.6	Conclusion . . . . .	106
<b>6</b>	<b>Raw Sequencing Key Value Model</b>	<b>107</b>
6.1	Introduction . . . . .	107
6.2	Background . . . . .	110
6.2.1	SAM . . . . .	110

---

6.2.2	BAM . . . . .	112
6.3	Key Value Model . . . . .	114
6.3.1	Subject table . . . . .	117
6.3.2	Position Table . . . . .	118
6.3.3	Cross-study Table . . . . .	120
6.4	Performance Evaluation . . . . .	121
6.4.1	Test environment and datasets . . . . .	121
6.4.2	Query over alignments overlapping specified regions . . . . .	122
6.5	Integration into tranSMART . . . . .	125
6.6	Discussion . . . . .	126
6.6.1	Family organization . . . . .	126
6.6.2	Extra disk space . . . . .	127
6.6.3	Long MML binning schema . . . . .	128
6.7	Conclusion . . . . .	131
<b>7</b>	<b>Gene expression based patient stratification application</b>	<b>132</b>
7.1	Introduction . . . . .	132
7.2	Background . . . . .	138
7.2.1	Correlation Algorithms . . . . .	138
7.2.2	Parallel R packages . . . . .	138
7.3	Parallel R approaches . . . . .	140
7.3.1	Statement level parallelizing . . . . .	140

7.3.2	Data level parallelizing . . . . .	142
7.4	Conclusion . . . . .	152
<b>8</b>	<b>Conclusion</b>	<b>156</b>
8.1	Summary of Thesis Achievements . . . . .	156
8.1.1	Generic -omics big data model . . . . .	156
8.1.2	Microarray Key Value model . . . . .	156
8.1.3	SNP Key Value model . . . . .	157
8.1.4	Raw Sequencing Key Value model . . . . .	157
8.1.5	Gene expression based patient stratification application . . . . .	157
8.2	Future Work . . . . .	158
8.2.1	Microarray Model . . . . .	158
8.2.2	SNP Model . . . . .	158
8.2.3	Raw Sequencing Model . . . . .	158
8.2.4	Coprocessors . . . . .	159
	<b>Bibliography</b>	<b>160</b>



# List of Tables

3.1	Subject table . . . . .	38
3.2	Position table . . . . .	39
3.3	Cross-study table . . . . .	41
4.1	TranSMART microarray data model. PROBESET_ID represents a genetic identity, SUBJECT_ID is a subject identity, STUDY_NAME shows a study identity, and RAW, LOG and ZSCORE are three types of expression values. . . . .	49
4.2	Example of a relational model representation of a patient record . . . . .	51
4.3	Key Value Microarray Subject Table schema. . . . .	53
4.4	Example in DEAPP table of tranSMART . . . . .	54
4.5	Data sample from LOG Family HFile . . . . .	55
4.6	Data sample from RAW Family HFile . . . . .	55
4.7	Data sample from RAW Family HFile . . . . .	55
4.8	Starting key for Key Value query . . . . .	55
4.9	Query result Key1 . . . . .	56
4.10	Query result Key2 . . . . .	56

4.11	Microarray position table . . . . .	58
4.12	Microarray Cross-study table . . . . .	58
5.1	TranSMART SNP_INFO table . . . . .	87
5.2	TranSMART VARIANT_SUBJECT_DETAIL table . . . . .	87
5.3	TranSMART SNP Subject table . . . . .	91
5.4	TranSMART SNP position table . . . . .	92
5.5	TranSMART SNP Cross-study table . . . . .	94
6.1	BAM Index . . . . .	114
6.2	Alignment Key Value Subject table . . . . .	117
6.3	Alignment Key Value Position table . . . . .	119
6.4	Alignment Key Value Cross-study table . . . . .	121
6.5	Alignment Key Value Position END table . . . . .	129
6.6	Alignment Key Value Cross-subject END table . . . . .	130

# List of Figures

2.1	The overview of tranSMART. The left side shows an ontology tree view of clinical data. The right side contains two subsets. Each subset has multiple selection conditions. Samples within a condition box are combined together. The final subset equals the samples shared by all conditions boxes.	14
2.2	The graphic view of clinical data.	15
2.3	The table view of clinical data.	15
2.4	The high dimensional parameter setting. This is the Heatmap function in the Analysis section. Cohorts indicates the two subsets to be compared. Variable Selection shows which high dimensional dataset is used for this function. Pathway Selection shows which genes are considered. If no gene specified, all probes in this dataset will be loaded.	16
2.5	The heatmap view of high dimensional data.	16
2.6	The Data Export service in tranSMART. There are several multi-option check boxes in the table. Each box indicates one dataset in a subset.	17
2.7	The Export Job service in tranSMART. The table indicates all the export jobs for the user.	17
2.8	The architecture of original tranSMART.	18
2.9	The analysis workflow in current tranSMART.	19

2.10	The architecture of new tranSMART. Three layers: the bottom layer is the data storage layer, the middle layer is the functional layer and the top layer is the visualization layer. . . . .	19
2.11	The high dimensional data query flow of the new tranSMART. . . . .	20
2.12	HBase architecture . . . . .	23
2.13	HFile structure . . . . .	24
2.14	the structure of the Key Value pairs . . . . .	24
2.15	HBase query example . . . . .	26
2.16	CCIndex architecture [ZLW <sup>+</sup> 10] . . . . .	27
2.17	The architecture of MongoDB sharding. [Cho13] . . . . .	28
3.1	Traditional molecular profiling data model . . . . .	33
3.2	CGDM structure. . . . .	34
3.3	Structure of multiple types of data. . . . .	35
3.4	Generic data model. StudyID is the identity of clinical trial or the name of a dataset, SubjectID is the identity of a subject in an assay, PositionID is a gene or protein location expressed in an assay and Others is additional information. . . . .	37
4.1	I2B2 data model [MWM <sup>+</sup> 10] . . . . .	46
4.2	I2B2 example . . . . .	47
4.3	tranSMART example . . . . .	48
4.4	Example of a JSON object that maps to the patient record illustrated in Table 4.2. The ”_id” is automatically generated by MongoDB to distribute JSON data. . . . .	51

---

4.5	Microarray Key Value model . . . . .	53
4.6	Medical use cases . . . . .	61
4.7	Performance of Key Value vs. original tranSMART application. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries. . . . .	62
4.8	Performance of Key Value in HBase vs. relational data model in HBase, MySQL Cluster and MongoDB. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries. . . . .	63
4.9	Performance of Random Read vs. Scan in Key Value data model. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries using both Random Read and Scan methods. The numbers above scan bar show the patient numbers the scan read in that case. The error bar shows the deviation of each test. . . . .	64
4.10	Performance of Random Read in different Families. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries of three Families. The error bar shows the deviation of each test. . . . .	65
4.11	Performance of Scan in different Families. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries of three Families. The error bar shows the deviation of each test. . . . .	65
4.12	Query by position within one dataset. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries. The error bar shows the deviation of each test. . . . .	67
4.13	Query by position across three datasets. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries. The error bar shows the deviation of each test. . . . .	69

4.14	Pseudocode of calling microarray data query functions. . . . .	69
4.15	Pseudocode of retrieving microarray data from HBase. . . . .	70
4.16	Data export service in tranSMART (select patient set). . . . .	71
4.17	Data export service in tranSMART. (select high dimensional data set). . .	72
4.18	U-BIOPRED dataset structure. . . . .	73
4.19	tranSMART instance with U-BIOPRED data. . . . .	74
4.20	Performance evaluation on U-BIOPRED data. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries. The bottom line shows the name of the item in the clinical tree in Figure 4.19 . . . . .	75
5.1	dbSNP data model . . . . .	83
5.2	VCF data format . . . . .	84
5.3	HapMap data format . . . . .	86
5.4	SeqWare Query Engine schema. The main table stores multiple genome information, such as generic features, variants and coverages. Each genome is represented by a particular Column Key (such as variant:genome6). Secondary indexing is implemented using a secondary table per genome indexed. The key is the column being indexed with the ID of the interest object. The value is represented by the row key in the main table. [OMN10]	88
5.5	SNP Key Value data model. . . . .	89
5.6	HapMap SNP data sample. There are two tables in this schema. The top table stores general SNP information. The bottom table stores sample and observation data. . . . .	90
5.7	Range scan by position within one study. . . . .	97

---

5.8	Random read by position within one study. . . . .	97
5.9	Query by subject. The Key Value bar represents the Key Value model on HBase. All the other bars represents the relational model on a specific database. . . . .	98
5.10	Random read 1 subject. The Key Value bar represents the Key Value model on HBase. All the other bars represents the relational model on a specific database. . . . .	99
5.11	Query over one study. . . . .	100
5.12	Query over multiple studies. . . . .	100
5.13	Random read 100 SNPs. The Key Value bar represents the Key Value model on HBase. The Key Value tests in left three groups use the Position table and the one in the last group uses the Cross-study table. All the other bars represents the relational model on a specific database. . . . .	101
5.14	Concurrent random read 100 SNPs. The Key Value curve represents the Key Value model on HBase using the Cross-study table. All the other curves represents the relational model on a specific database. . . . .	102
5.15	The pseudocode of variant data query by subject. . . . .	102
5.16	The pseudocode of variant data query by position within a study. . . . .	103
5.17	The pseudocode of variant data query by position across studies. . . . .	103
5.18	Performance evaluation on tranSMART. . . . .	105
6.1	SAM data sample . . . . .	113
6.2	Raw sequencing Key Value model. . . . .	116
6.3	Alignment query using less than 10 threads. The region size is less than 100,000 bp. . . . .	124

- 
- 6.4 Alignment query using more than 50 threads. The maximum region size is 500,000 bp. . . . . 124
- 6.5 Alignment query with different Families. The maximum thread number is 100 and the maximum region size is 500,000 bp. . . . . 125
- 6.6 Scan overlapping a specific region within a subject. . . . . 126
- 6.7 Scan alignment records overlapping a specific region across subjects. . . . 126
- 6.8 Random read alignment records by RNAME. . . . . 127
- 7.1 Cohorts selection of severe samples. In subset1 there are two condition boxes. The first box contains all CD4+ T-cells and the second box contains all severe samples. Thus, the subset1 contains all the samples who are both CD4+ T-cells and severe. Then, the subset2 contains all samples who are both CD8+ T-cells and severe. . . . . 133
- 7.2 Hierarchical Clustering parameters. The cohorts are set in Figure 7.1. No gene is specified. So all probe data are loaded, but only the first 50 probes are shown in the heatmap. Clustering is applied for both rows and columns. 134
- 7.3 Hierarchical Clustering result of severe samples. The first 8 columns are samples in subset1 and the remaining 8 columns are samples in subset2. According to the clustering by column shown at the top of the graph, two subset are exactly divided. . . . . 135
- 7.4 Cohorts selection of not severe samples. In subset1 there are two condition boxes. The first box contains all CD4+ T-cells and the second box contains all healthy and non-severe samples. Thus, the subset1 contains all the samples who are healthy and non-severe CD4+ T-cells. Then, the subset2 contains all samples who are healthy and non-severe CD8+ T-cells. 136



- 
- 7.5 Hierarchical Clustering result of severe samples. The first 10 columns and 21st and 23rd columns are samples in subset1 and the remaining 12 columns are samples in subset2. According to the clustering by column shown at the top of the graph, two subset are not divided into two clusters. 137
- 7.6 Parallelizing R nps function. The grep part on the left is the for loop code to be parallelized. The code within the brackets is node changed. The new function gaploop has a similar parameter i. This new function can be used by R parallel packages, such as Snow or Snowfall. The sfLapply is a function in the Snowfall package. . . . . 141
- 7.7 Statement level parallel method evaluation. . . . . 142
- 7.8 Basic correlation matrix calculation using MapReduce . . . . . 143
- 7.9 Correlation matrix calculation with redundant coefficient calculations skipped 144
- 7.10 Matrix transformation. The elements in the bottom left triangle are mapped to the top right triangle. Each row represents a Reducers load, which contains either c elements or f elements. . . . . 145
- 7.11 Balanced reducers. . . . . 146
- 7.12 Calculation in Combiner. This is the MapReduce framework where all calculations in Reducers are moved to Combiner according to the algorithm in Figure 7.11. . . . . 147

7.13 Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method shows the data preparation time. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time. . . . . 150

7.14 Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method shows the data preparation time, including data retrieval from a relation database. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time. . . . . 151

7.15 Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method shows the data preparation time, including data retrieval from a Key Value database. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time. . . . . 152

7.16 This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method shows the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on ONCOLOGY dataset. . . . . 153

- 7.17 This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method shows the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on the cross-study consisting of ONCOLOGY and LEukemia (4254 subjects). . . . . 154
- 7.18 This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method shows the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on the large artificial dataset (8508 subjects). . . . . 155

# Chapter 1

## Introduction

### 1.1 Motivation and Objectives

Molecular profiling refers to the study of specific patterns or signatures, such as DNA polymorphism, mRNA gene expression profiling, RNA profiling, proteomic and metabolic polymorphism. High-throughput molecular profiling has greatly improved patient stratification and mechanistic understanding of disease [SF05]. Biomedical research is moving towards using high-throughput molecular profiling data to improve clinical decision making [Gar05]. TranSMART, a biomedical data warehouse and analytics software platform that integrates clinical and genomics data [ABHG13], enables clinicians to search and analyze these data by tools such as Gene pattern [RLG<sup>+</sup>06] and R [R C14]. Gene expression microarray [Qua01] and single nucleotide polymorphism (SNP) data have been frequently used in tranSMART, while raw mRNA sequence data is planned to be integrated.

However, in the big data era molecular profiling data size increases sharply due to new biological techniques. For example, next generation sequencing (NGS) [Met10] techniques create quantities of raw sequence and SNP data. The data warehouse in tranSMART cannot meet the three "V" requirements of big data (Volume, Variety, and Velocity). Fortu-

nately, technique revolutions not only happened in biology but also in computing. Emerging Key Value stores, such as Apache HBase [Geo08] and Google Bigtable [big08], can help solve the problem. But none of existing molecular data model can be easily re-used in new Key Value models due to the different database models.

Thus, the motivation for this thesis to create a generic -omics [ZLN10] big data model for molecular profiling data in order to optimize tranSMART data warehouse so that clinicians can make use of it for faster clinical decision making in big data age.

### **1.1.1 Microarray**

An increasing amount of biological data being produced has predicated the use of databases capable of storing and analyzing genomic expression data. In the field of translational research, knowledge management platforms use databases to store a variety of data produced from clinical studies, including patient information, clinical outcomes as well as high-dimensional -omics data. For optimal analyzes and meaningful interpretations, such databases also store legacy data taken from public sources, such as the Gene Expression Omnibus (GEO) [BWL<sup>+</sup>13] and the Gene Expression Atlas [KAB<sup>+</sup>12], alongside new study data. This enables cross-study comparisons and cross-validation to take place. High-throughput transcriptomic data generated by microarray experiments is the most abundant and frequently stored data type currently used in translational studies.

A typical query involves searching for and retrieving patient gene/protein expression levels in a study in order to perform data analysis. For each study, the transcriptomic data stored in the database is comprised of individual probe set values for each patient sample (in some cases multiple samples from each patient are profiled), and annotation information further describing the experiment (e, g. trial information, microarray platform identifier, normalization method, gene descriptions). Once retrieved the data is passed to analytical tools, such as GenePattern or custom R workflows for further analysis or

visualization.

With the increasing amount of data used in translational medicine studies, especially in recent years, the performance of tranSMART in terms of data retrieval and statistical processing is decaying, if the statistical analyses enabled by tranSMART are to be used beyond translational research for clinical decision making. For example, high-throughput transcriptomic data generated by microarray experiments is the most abundant and frequently stored data type currently used in translational studies. For the needs of various collaborative translational research projects, an instance of tranSMART is hosted at Imperial College London and has been configured to use an Oracle relational database for back-end storage. It currently holds over 70 million gene expression records. When querying the database simultaneously for hundreds of patient gene or protein expression records, a typical exercise in translational studies, the record retrieval time can currently take up to more than 10 minutes. These kinds of response times impede analyzes performed by researchers using this deployed configuration of tranSMART. If high-throughput sequence data, such as SNP or raw sequence, generated by NGS technologies are considered, the total processing time becomes impractical.

### 1.1.2 SNP

Thousands of genome-wide association studies (GWAS) of significant human diseases have been conducted recently [SSR11]. Many of such studies use single nucleotide polymorphism (SNP) arrays [LaF09], for example in the International HapMap project [Con07, The10, The05] that aims to develop a map describing the common patterns of human genetic variation. SNP has already benefited identification of disease subgroups and the prediction of responses of individual subjects, named molecular profiling based patient stratification. Biomedical research is moving towards using high-throughput SNP data to improve clinical decision making. One approach for building clusters is to strat-

ify subjects based on their SNPs. Unsupervised clustering algorithms can be used for stratification purposes.

The motivation for optimization of SNP data model is based on the experiences in using tranSMART. The aim is to optimize tranSMART so that clinicians can make use of it for faster and more confident clinical decision making. However, the current tranSMART SNP query velocity is low. For example, when 137 subjects with 317,642 genotype pairs are retrieved from a standalone PostgreSQL database version 9.4 instance in a OpenStack Ubuntu virtual machine configured with 16 CPU cores and 16 GB of memory, the query takes several minutes. When total number of SNPs increases to around 3 million, the query time becomes unacceptable long. If thousands of such subjects are retrieved, the execute time will be impractical.

### **1.1.3 Raw Sequence**

RNA sequencing (RNA-seq) [MWM<sup>+</sup>08, WGS09] has been used for profiling of transcriptomes in many areas of biology, including studies in medicine and pharmacy. The purpose of this part is to improve clinical decision making through the discovery of differentially expressed genes across patient and control cases in a cohort. RNA-seq data processed by the protocol [AMC<sup>+</sup>13] is suitable for the molecular profiling based patient stratification function with sequence data.

High-throughput transcriptomic data generated by NGS technologies is the most popular data type currently used in translational studies. With its unprecedented throughput, scalability, and speed, NGS enables researchers to study biological systems at a level never before possible. For the needs of various collaborative translational research projects, tranSMART is starting to support NGS raw sequencing data for translational studies. But the current tranSMART was designed to host data in the gigabyte range only. Anticipating the requirement to store and analyze NGS data, where the volume of data being produced



will be in the terabyte range, the speed exhibited by tranSMART could be unacceptably slow.

A typical query involves searching for raw sequences overlapping a specific region for a gene or a trait to count its expression level and perform data analysis. The current tranSMART plans to use the binary version of sequence alignment mapping (SAM) files (BAM) to store raw sequencing data. Once retrieved the data is passed to analytical tools, such as Samtools [LHW<sup>+</sup>09], UCSC Genome Browser [KHK11] and custom R workflows for further analysis or visualization.

#### 1.1.4 Application

An application is added to show how these data models can be used for analysis functions used in tranSMART. One approach for molecular profiling based patient stratification is to stratify subjects based on their molecular profiles. Many machine learning algorithms can be used for stratification purposes, including supervised learning and unsupervised learning. Supervised learning will be applied if other features of patients have been known, such as patient cohorts. Unsupervised learning algorithms will be used if only molecular profiling data are gathered. Clustering methods are frequently used unsupervised learning methods in tranSMART, such as hierarchical clustering and k-means clustering. A hierarchical clustering application demonstrates how tranSMART can help patient stratification. The application used four kinds of correlation methods with high-dimensional molecular profiling data (gene expression data), by taking full advantage of a programming model specifically designed for parallel processing of big datasets using the Key Value data models.

However, the current tranSMART performance of preparing correlation matrices when analyzing high dimensional molecular data is sub-standard. For example, an unsupervised hierarchical clustering has been performed on the publicly available Multiple myeloma

dataset taken from the National Center for Biotechnology Information (NCBI) GEO. The dataset contains 559 subjects gene expression data produced by an Affymetrix GeneChip Human Genome U133 Plus 2.0 Array. In order to build the clusters, the subjects are clustered using a hierarchical clustering algorithm implemented in R. In preliminary tests, running this algorithm on a virtual machine configured with 32 cores and 32 GB of memory took over 6 minutes using a Euclidean distance matrix, and more than a week if performing a Kendall rank correlation. These observations show that the bottleneck in the hierarchical clustering algorithm was in generating the correlation matrix. Clearly optimizing the performance of these analyses would expedite clinical decision making. With high throughput sequencing technologies promising to produce even larger datasets per subject, the performance of the state-of-the-art statistical algorithms is to be further impacted unless efforts towards optimization are carried out. The paper [WPJ<sup>+</sup>14] describes a series of processing optimizations on the calculation of correlation matrices used for hierarchical clustering implemented on R Rhipe [GHR<sup>+</sup>12b] plugin, which works on Hadoop ecosystem. This optimization performed up to 16.56 times faster than vanilla R. The most important procedure in the whole parallel R workflow is data loading and distribution using the Key Value models.

## 1.2 Contributions

In this thesis, a Collaborative Genomic Data Model (CGDM) has been introduced to solve all these issues in the previous Motivation section. CGDM creates three Collaborative Global Clustering Index Tables (CGCITs) to solve the velocity and variety issues at the cost of limited extra volume. Three instances based on CGDM Key Value [SULS09] data model, including microarray data model, SNP data model and raw sequencing data model, have been implemented and fully evaluated. Finally, an application demonstrates how CGDM benefits the performance.

The microarray Key Value model focuses on resolving the velocity problems in gene expression microarray data processing, due to the increasing volume of data being produced. The experiments show that, the microarray Key Value implementation of CGDM using HBase outperforms the tranSMART relational data model on HBase, MySQL Cluster and MongoDB by up to 246, 7 and 20 times. When this Key Value implementation integrated into tranSMART, it performs up to 7 times faster than the current tranSMART.

The SNP Key Value model is aimed at solving the velocity issues faced in mutation databases, due to the increasing volume of data being produced by next-generate sequencing techniques. The performance tests indicate that the SNP Key Value implementation of CGDM using HBase performs up to 351 and 9 times faster than the tranSMART SNP relational model on HBase and MySQL Cluster. When integrated into tranSMART this Key Value implementation performs up to 7 times faster than the current tranSMART.

The raw sequencing Key Value model is aimed at solving the performance issues faced in gene sequence databases, due to the increasing volume of raw sequencing data being produced high-throughput short sequencing techniques. The results illustrate that this implementation using HBase outperforms the sequence alignment mapping (SAM) model using HBase and a BAM server by up to 440 and 22 times.

Through the hierarchical clustering application, the CGDM microarray implementation reduces 6/7 of time in the data preparation procedures. The application demonstrates CGDM holds great promise for large data analysis with the data size increasing.

## 1.3 Organization

Chapter 2 introduces the current tranSMART architecture and the prospective one will be first introduced. Then the new techniques, HBase and CCIndex, used to implement the prospective architecture will be described.

Chapter 3 presents optimisations to the database schemas used in tranSMART for molecular profiling data that will significantly improve the of Input/Output performance in tranSMART.

Chapter 4 explains a Key Value based data model to support fast queries over large-scale microarray data, and implemented the schema using HBase, compared the query retrieval time against the traditional relational data model running on HBase, MySQL Cluster and MongoDB.

Chapter 5 proposes a new Key Value SNP model with a supplementary cluster index to overcome the disadvantages resulting from secondary index and provide fast query on all indexed properties.

Chapter 6 proposes a raw sequencing Key Value model in order to provide fast query over raw sequencing data.

Chapter 7 presents an application using different types of data stored in the models above to evaluate the contributions of the data models.

Finally, Chapter 8 summarizes the thesis and presents ideas for further works.

## **1.4 Statement of Originality**

I declare that the content of thesis is composed by myself, and the work it presents is my own. All use of the previously published work of others has been listed in the bibliography.

## 1.5 Publications

- **High Dimensional Biological Data Retrieval Optimisation with NoSQL Technology.** In: *BMC Genomics supplement*. [WPW<sup>+</sup>14].

This paper presents a new data model better suited for high-dimensional data storage and querying, optimized for database scalability and performance. A new Key Value pair data model has been designed to support faster queries over large-scale microarray data and implemented the model using HBase, an implementation of Google's BigTable storage system. An experimental performance comparison was carried out against the traditional relational data model implemented in both MySQL Cluster and MongoDB, using a large publicly available transcriptomic data set taken from NCBI GEO concerning Multiple Myeloma. Our new key-value data model implemented on HBase exhibits an average 5.24-fold increase in high-dimensional biological data query performance compared to the relational model implemented on MySQL Cluster, and an average 6.47-fold increase on query performance on MongoDB.

- **Optimizing parallel R correlation matrix calculations on gene expression data using MapReduce.** In: *BMC Bioinformatics*. [WPJ<sup>+</sup>14].

This paper evaluates the current parallel modes for correlation calculation methods and introduces an efficient data distribution and parallel calculation algorithm based on MapReduce to optimise the correlation calculation. The performance has been studied using two gene expression benchmarks. In the micro-benchmark, the new implementation using MapReduce, based on the R package RHIPE, demonstrates a 3.26-5.83 fold increase compared to the default Snowfall and 1.56-1.64 fold increase compared to the basic RHIPE in the Euclidean, Pearson and Spearman correlations. Though vanilla R and the optimised Snowfall outperforms the optimised RHIPE in the micro-benchmark, they do not scale well with the macro-

benchmark. In the macro-benchmark the optimised RHIPE performs 2.03-16.56 times faster than vanilla R. Benefiting from the 3.30-5.13 times faster data preparation, the optimised RHIPE performs 1.22-1.71 times faster than the optimised Snowfall. Both the optimised RHIPE and the optimised Snowfall successfully performs the Kendall correlation with TCGA dataset within 7 hours. Both of them conduct more than 30 times faster than the estimated vanilla R.

- **DSIMBench: A Benchmark for Microarray Data Using R.** In: *The 5th workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware. Hangzhou, China, 2014.* [WPE<sup>+</sup>14].

Parallel computing in R has been widely used to analyze microarray data. We have seen various applications using various data distribution and calculation approaches. Newer data storage systems, such as MySQL Cluster and HBase, have been proposed for R data storage; while the parallel computation frameworks, including MPI and MapReduce, have been applied to R computation. Thus, it is difficult to understand the whole analysis workflows for which the tool kits are suited for a specific environment. This paper proposes DSIMBench, a benchmark containing two classic microarray analysis functions with eight different parallel R workflows, and evaluate the benchmark in the IC Cloud testbed platform.

# Chapter 2

## Background

In this Chapter, the current tranSMART architecture and the prospective one will be first introduced. Then the new techniques, HBase and CCIndex, used to implement the prospective architecture will be described.

### 2.1 tranSMART architecture

#### 2.1.1 Overview

Originally developed by Johnson and Johnson for internal knowledge management in translational studies, tranSMART, a translational medicine data warehouse, has recently been open-sourced in github (<https://github.com/transmart>). It consists of a data repository and a dataset explorer. The data repository allows the storage of a range of clinical study data such as genotypic data, demographics, clinical observations, clinical trial outcomes, biomarker data, annotated with information about the datasets, from open and commercial databases. The dataset explorer enables search and analyses, including connecting to tools such as Gene pattern and R.

Translational Research studies are characterized by a span of data types that ranges from descriptive clinical and patient data to highly dimensional molecular profiling data. The tranSMART repository presents a study based view of the data to provide provenance and evidence support for analysis workflows and study findings [SKKP10]. Particularly, the idea of an evidence repository expands the scope of a study management database to include support for any collaborative research project. Users can upload their datasets or select datasets from other studies to integrate for complex analysis. Both public and private studies, as well as other user-defined research projects, can be stored in this repository.

The tranSMART platform provides tools for loading a variety of data types, such as clinical and -omics research data, from a range of sources. The platform enables scientists to discover and refine hypotheses by investigating the correlation between genetic and genomic data for cohorts of patients, and to assess their analytic results in the context of published literature and their internal work. It was designed to meet the needs of general scientists, clinician-scientists, pharmacologists, toxicologists, and others, who are involved in therapeutics and diagnostics discovery and development of new therapies for patients.

The tranSMART platform implements a set of data models, data transformation utilities, and analytical web applications all centered on the trial, to accelerate discoveries by creating a standardized and semantically integrated data warehouse of research results linked to reusable and scalable self-service analytics.

An API has been added to the tranSMART platform to allow connection to external packages for extended data viewing, such as Genome Browser, statistical analysis based on R language and Spotfire [Ahl96] visualization and analytics. These may be part of the distributed open source platform or distributed privately by vendors who are using the open API.



For example, in Figure 2.1 the Navigate Terms in the left side shows the clinical data view. Each item contains several sample data. The GPL570 under Biomarker Data contains 40 high dimensional data blocks; the CD4 under Cell Type contains 20 character records; The FEV1 under Clinical contains 40 numeric records. Each high dimensional data block contains genomic expression or genetic variant records for one sample, such as RNA microarray and SNP data. In this example, the GPL570 means this assay used an Affymetrix HG-U133 chip and all the probe information can be looked up in the chip instruction document. The right subset boxes are conditions used to select samples. In this example subset1 contains healthy samples only, whilst subset2 contains severe samples. Clinical data can be showed in either a graph view in Figure 2.2 or a data table in Figure 2.3. High dimensional data can be viewed in Heatmap Analysis in Figure 2.4, where high dimensional item and gene filters are set, and in Figure 2.5. In this example, GPL570 item and gene DDR1 are selected. Other advanced analysis functions, such as hierarchical clustering and marker selection, can also be applied to high dimensional data. The Data Export service offers off-line raw clinical and high dimensional data exportation, as shown in Figure 2.6. If a Data Export job is started, there will be a job item in the Export Job page, as shown in Figure 2.7.

### **2.1.2 Data model**

The original tranSMART provided a data model for clinical data based solely on Informatics for Integrating Biology & the Bedside (I2B2) [MWM<sup>+</sup>10], as shown in Figure 2.8, a generic data model designed to support flexibility for cohort analysis and discovery. There are three layers in this architecture. The bottom layer is the data storage layer, the middle layer is the functional layer and the top layer is the visualization layer. Most of the data are stored in a relational database, such as Oracle or PostgreSQL; document-oriented data are stored in a Apache Solr [MHG10] system; temporary data are stored in a local file

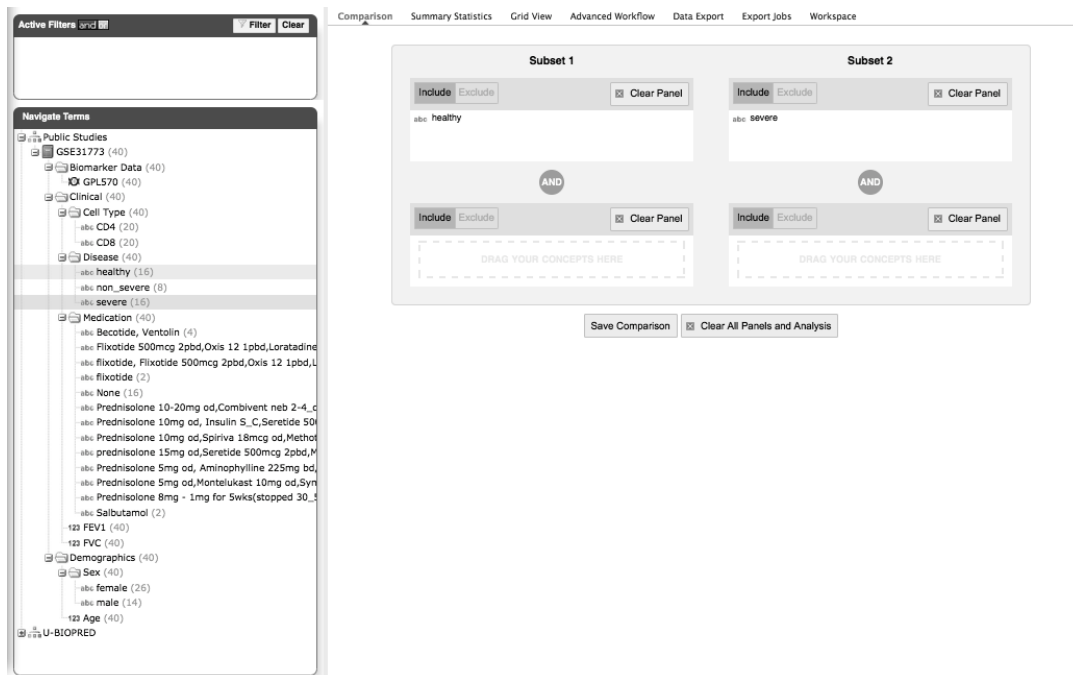


Figure 2.1: The overview of tranSMART. The left side shows an ontology tree view of clinical data. The right side contains two subsets. Each subset has multiple selection conditions. Samples within a condition box are combined together. The final subset equals the samples shared by all conditions boxes.

system, such as EXT3 or EXT4. TranSMART added a format to define study designs and attributes, which is significant for cross-study analysis [GMPZ<sup>+</sup>08]. This study concept enables this data model to support data updates, live collaborations on studies as well as cross-talk and data sharing between studies.

There are two general categories of data types in translational research studies: clinical and patient data and genomic characterization data produced from -omics technologies [SRH14]. In tranSMART data warehouse, data are organized as a tree and each clinical or -omics measurement is a node in the tree. The name of the node called a concept is brief description of the measurement. A clinical concept contains only one feature. For example, the concept FEV1 contains the value of forced expiratory volume in 1 second. An -omics concept usually contains multiple features. The feature number relies on the -omics experiment. For instance, the concept GPL570 (Affymetrix Human Genome U133 Plus 2.0 Array) usually includes 54,675 features. Thus, the clinical data are also called

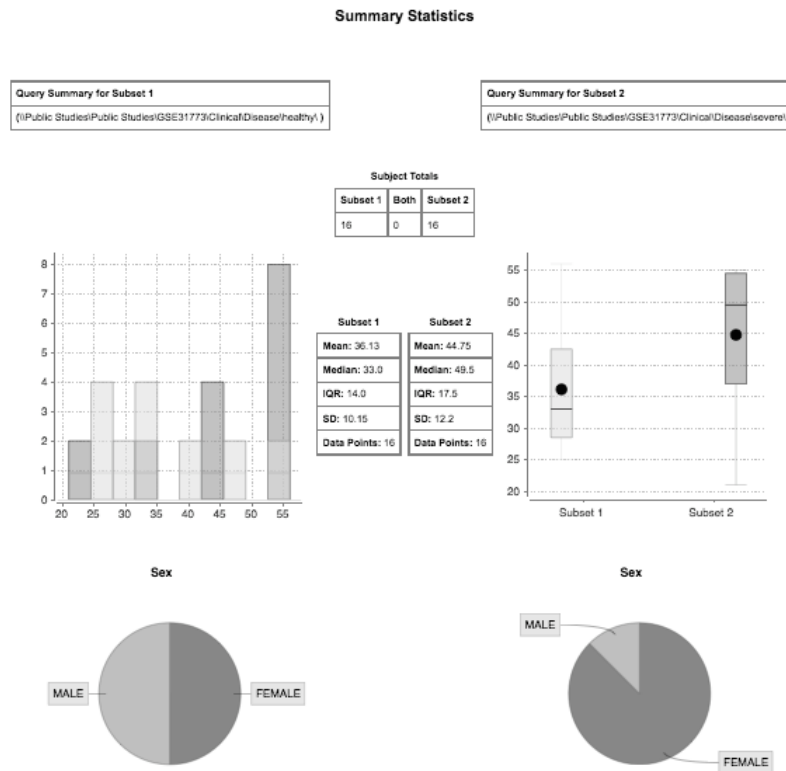


Figure 2.2: The graphic view of clinical data.

Grid View									
Subject	Patient	Samples	Subset	Trial	Sex	Age	Race	severe	healthy
1000385492	11		subset1	GSE31773	female	31	NULL	NULL	healthy
1000385494	34		subset1	GSE31773	female	25	NULL	NULL	healthy
1000385495	38		subset1	GSE31773	female	33	NULL	NULL	healthy
1000385502	36		subset1	GSE31773	male	46	NULL	NULL	healthy
1000385503	37		subset1	GSE31773	male	33	NULL	NULL	healthy
1000385504	35		subset1	GSE31773	female	26	NULL	NULL	healthy
1000385505	6		subset1	GSE31773	female	25	NULL	NULL	healthy
1000385506	10		subset1	GSE31773	female	33	NULL	NULL	healthy
1000385510	12		subset1	GSE31773	male	56	NULL	NULL	healthy
1000385514	8		subset1	GSE31773	male	46	NULL	NULL	healthy
1000385516	7		subset1	GSE31773	female	26	NULL	NULL	healthy
1000385519	9		subset1	GSE31773	male	33	NULL	NULL	healthy
1000385522	39		subset1	GSE31773	female	31	NULL	NULL	healthy
1000385524	5		subset1	GSE31773	male	39	NULL	NULL	healthy
1000385525	33		subset1	GSE31773	male	39	NULL	NULL	healthy
1000385527	40		subset1	GSE31773	male	56	NULL	NULL	healthy
1000385497	14		subset2	GSE31773	female	21	NULL	severe	NULL
1000385499	30		subset2	GSE31773	female	54	NULL	severe	NULL
1000385501	31		subset2	GSE31773	female	45	NULL	severe	NULL
1000385508	25		subset2	GSE31773	female	55	NULL	severe	NULL
1000385509	32		subset2	GSE31773	female	42	NULL	severe	NULL
1000385512	27		subset2	GSE31773	male	32	NULL	severe	NULL
1000385513	19		subset2	GSE31773	female	45	NULL	severe	NULL
1000385515	17		subset2	GSE31773	female	54	NULL	severe	NULL
1000385517	15		subset2	GSE31773	male	32	NULL	severe	NULL

Figure 2.3: The table view of clinical data.

low dimension data in tranSMART and the -omics are named high dimensional. Data from both categories are organized and defined with respect to the study in order to pro-

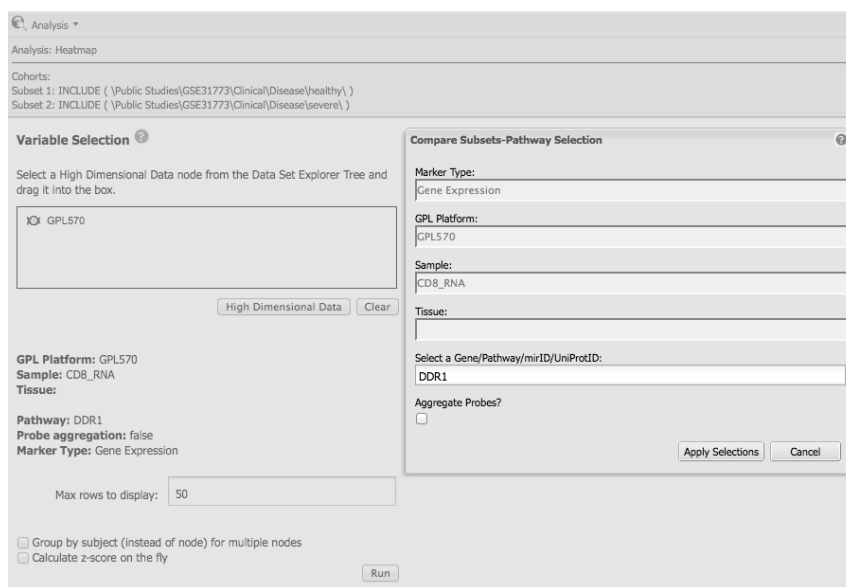


Figure 2.4: The high dimensional parameter setting. This is the Heatmap function in the Analysis section. Cohorts indicate the two subsets to be compared. Variable Selection shows which high dimensional dataset is used for this function. Pathway Selection shows which genes are considered. If no gene specified, all probes in this dataset will be loaded.

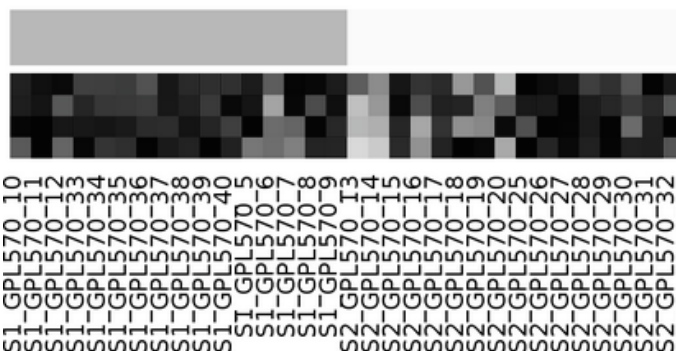


Figure 2.5: The heatmap view of high dimensional data.

vide the context for generating and analyzing such data. The study concept is the central one, which stores metadata about the study, such as design, purpose, hypotheses, etc. The low dimensional data forms the basis for clinical data and patient observations. Other hierarchical concepts could be defined based on the clinical concept such as cohorts, populations and measurements. The high dimensional data forms the basis for molecular profiling information.

TransSMART studies use different technologies to generate high dimensional profiling

	Subset 1	Subset 2
<b>Instructions:</b> 1. Select the check boxes to indicate the data types and file formats that are desired for export. 2. Optionally you can filter the data by dragging and dropping some criteria onto each data type row. 3. Click on the "Export Data" button at the bottom of the screen to initiate an asynchronous data download job. 4. To download your data navigate to the "Export Jobs" tab.		
Selected Cohort	INCLUDE ( \Public Studies\GSE31773\Cli...	INCLUDE ( \Public Studies\GSE31773\Cli...
Clinical & Low Dimensional Biomarker Data (Drag and drop low dimensional nodes he...	Data is available for 16 patients Export (.TXT) <input type="checkbox"/>	Data is available for 16 patients Export (.TXT) <input type="checkbox"/>
	Metadata will be downloaded in a separat...	Metadata will be downloaded in a separat...
Messenger RNA data (Microarray) (Drag and drop low dimensional nodes he...	Tab separated file is available for GPL570: 16 patients Export (.TSV) <input type="checkbox"/>	Tab separated file is available for GPL570: 16 patients Export (.TSV) <input type="checkbox"/>
	Metadata will be downloaded in a separat...	Metadata will be downloaded in a separat...

Figure 2.6: The Data Export service in tranSMART. There are several multi-option check boxes in the table. Each box indicates one dataset in a subset.

Name	Query Summary	Status	Started On
<a href="#">ShicaiWang-DataExport-109850</a>		Completed	

Figure 2.7: The Export Job service in tranSMART. The table indicates all the export jobs for the user.

data, such as genetic and proteomic microarray matrices, genotype and phenotype Variant Call Format (VCF) files and SAM files. These data are served either for analysis and visualization tools, or for users directly querying interpreted data and general summaries of study findings.

The current analysis workflow contains four steps, job configuration, data preparation, R analysis and result, as shown in Figure 2.9. The data preparation contains two sub-steps, database query and data format; R analysis includes loading data into R engine and performing analysis functions.

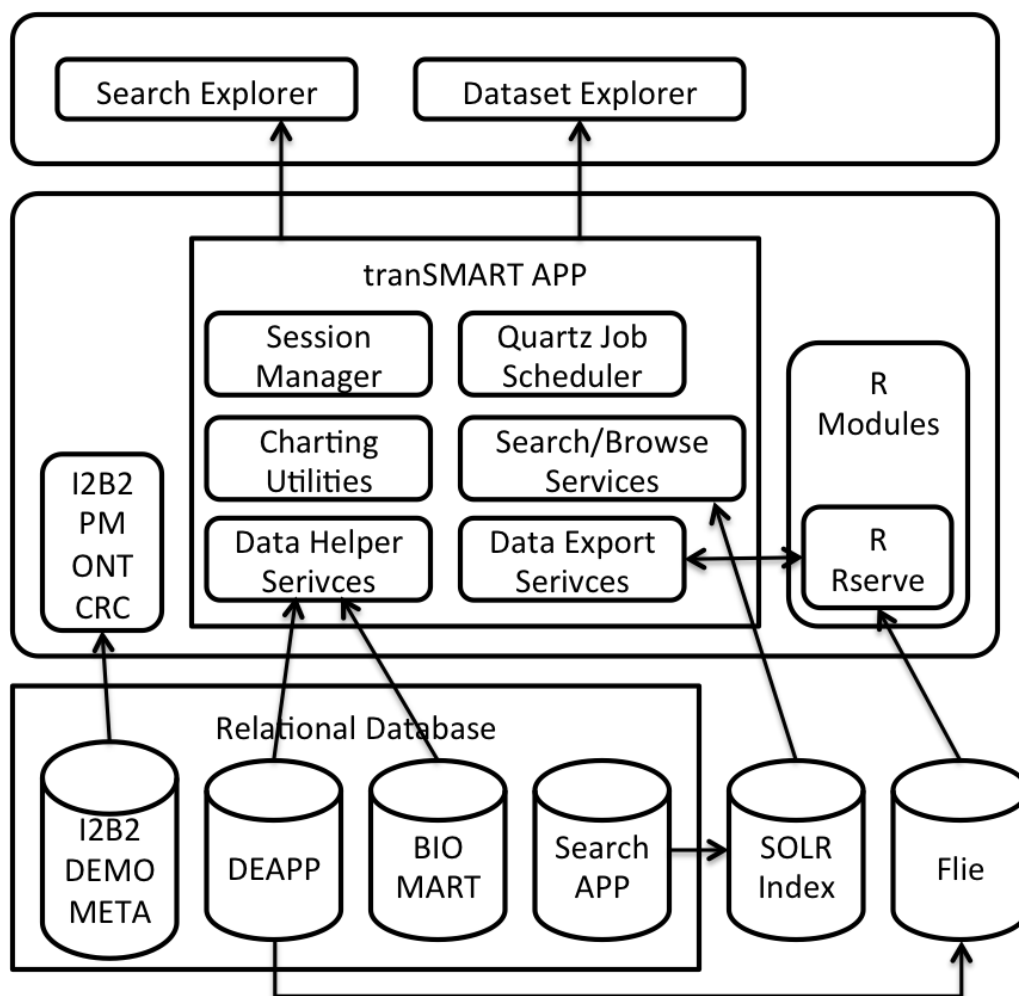


Figure 2.8: The architecture of original transSMART.

### 2.1.3 Future transSMART

With the high dimensional data size sharply increasing, the transSMART response time of data query and analysis became unacceptable long. Fortunately, technical revolutions not only happened in biology but also in computing. State of art distributed data storage engines and parallel analysis frameworks have been planned to be integrated into transSMART to resolve this problem, as shown in Figure 2.10. Two significant changes will be made in the high dimensional search engine and advanced analysis workflow.

High performance database clusters will replace the current standalone database to host high dimension data for high speed data queries. Parallel SQL database clusters, such as

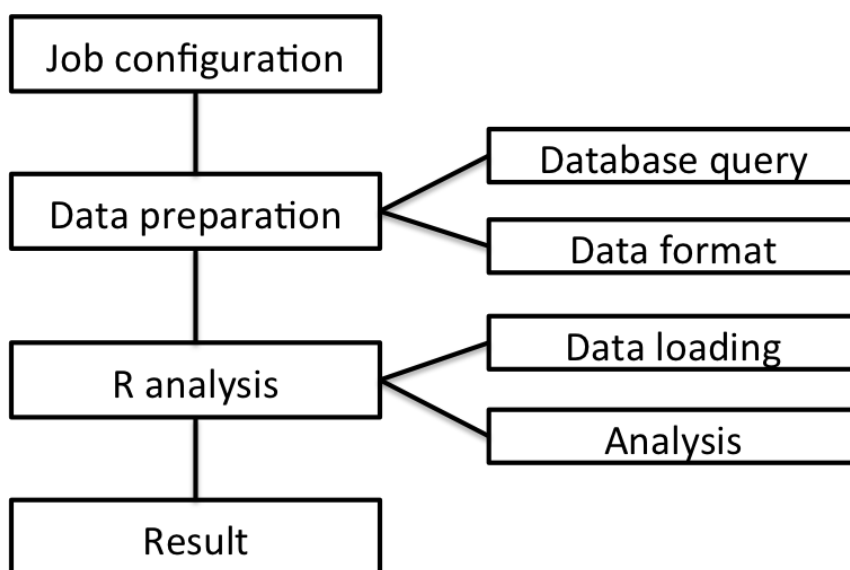


Figure 2.9: The analysis workflow in current tranSMART.

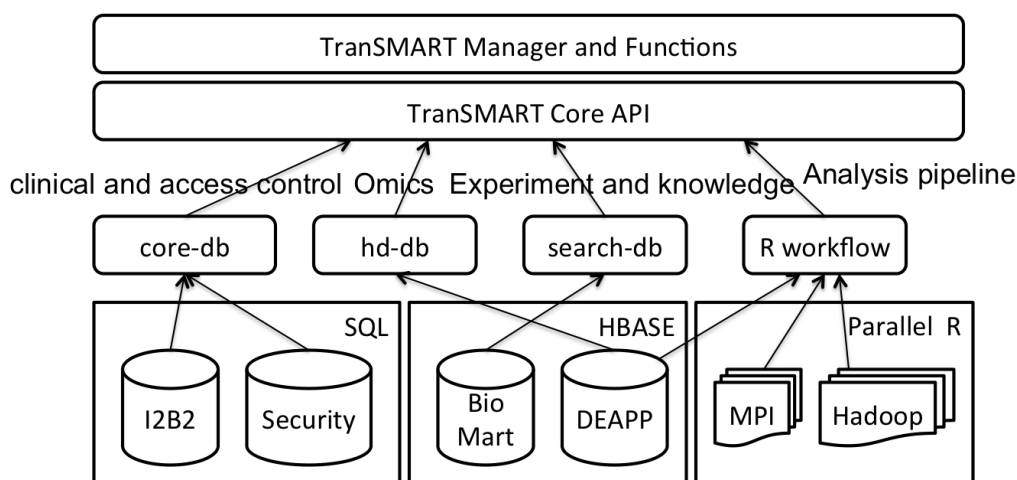


Figure 2.10: The architecture of new tranSMART. Three layers: the bottom layer is the data storage layer, the middle layer is the functional layer and the top layer is the visualization layer.

MySQL Cluster [RT04], PostgreSQL XE [Mom01] and SQL Server [Del00] are conventional all-purpose solutions. But new Key Value data stores can perform better for some specific usages. For example, HBase [Geo08] can offer an extremely fast range query service.

The choice of technology will be based entirely on the performance and accessibility to popular data analysis environments and visualization tools. The high dimensional data

query workflow is an important and time consuming procedure in the analysis workflow, as shown in Figure 2.11. The clinical data and patient identity information are still stored in a SQL-based database, while high dimensional data, such as genomic expression and genetic variant call, are stored in a NoSQL storage system. In this query flow, query conditions are loaded in the data query boxes and translated into SQL query conditions. With this query information, related patient identities and heredity information are retrieved to generate NoSQL high dimensional data query requests. Finally, the expected high dimensional data are retrieved from NoSQL databases.

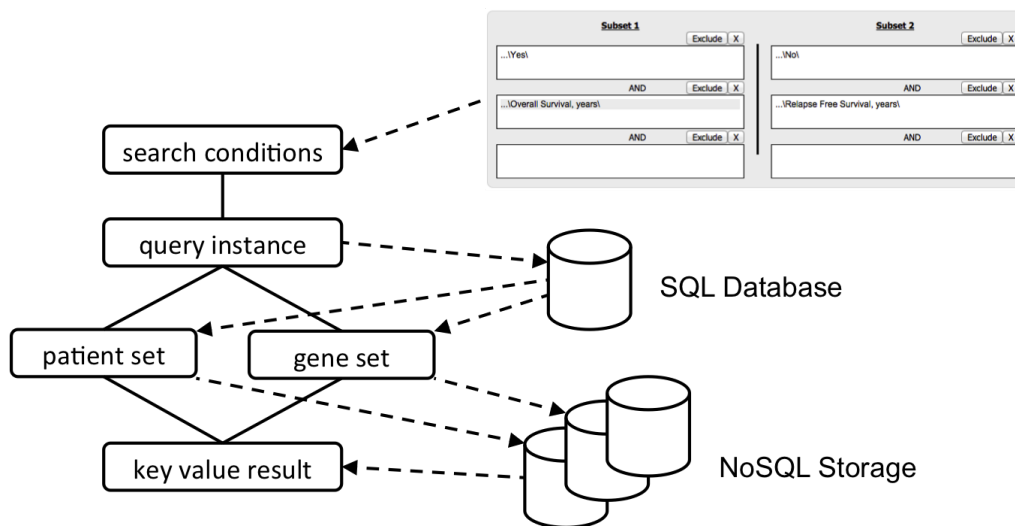


Figure 2.11: The high dimensional data query flow of the new tranSMART.

In addition, tranSMART infrastructure for open analytics is built around an extensible design. A set of APIs will be provided to enable users to build applications or write scripts to analyze project data stored in the repository. All these features are available via statistical environments such as R as well as compiled languages such as Java. Users will be able to either use these APIs to integrate with existing applications or create new analysis and integrate them with third party applications. In addition to the APIs, tranSMART hosts a range of common analysis and visualization tools based on R, such as cohort analysis [Pop72, GCB<sup>+</sup>04], differential gene expression analysis [RMS10, And10] and data exploratory visualization tools. This analytic platform will host a compute service that provides its users with remote access to R running on a R cluster. A similar setup is



hosted at Revolution Analytics, known as the Revolution R or RHadoop [OD14], which provides R built for multicore CPUs, access to a powerful computing infrastructure. This setup will allow fast access to data stored in the repository and a more efficient procedure for sharing workflows within a common environment.

## 2.2 Big Data Management

Big data is a common term used to describe the rapid growth and availability of large amounts of structured and unstructured data. Some of the current and popular examples of big data are the data from the NCBI GEO [citegeo], HapMap [citehapmap] and 1000 Genomes [Siv08]. Big data management is the process of storing, querying and analyzing these large and complex data collections. This section explores research work aimed towards addressing the main challenges of volume, velocity and variety in Big data. Volume refers to the enormous amount of data that are provided by many sources; velocity means acceptable response time for large data set queries; variety indicates the different data format and how to abstract an model fit for various types of data.

These high volumes of data result in the issue of providing a data model for data management that is scalable and efficient. For example, the model must be capable of efficiently storing and retrieving large volumes of molecular profiling information. It must also have the capacity to scale in order to handle incoming large number of high-throughput data.

Google Bigtable [big08] is a widely used (e.g. Google Scholar, Google Earth, etc.) database to store such large volumes of structured data in the range of petabytes across hundreds or thousands of machines, and to make it easy to add more machines to the system and automatically start taking advantage of those resources without any reconfiguration. The data model of Bigtable is a set of processors known as clusters. Each cluster controls a set of table partitions. A table in Bigtable is a sparse, distributed, persistent and

dynamic multidimensional sorted tree, known as Log-structured merge-tree (LSM-tree) [OCGO96], and the data is organized into three dimensions: rows, columns, and timestamps. Many open source projects are implemented based on the Bigtable design, such as Accumulo [SFG13], Cassandra [LM10], HBase [Geo08], Hypertable [KG06], Druid [YTL<sup>+</sup>14] and Open Neptune [AD11].

However, range queries by non-key columns can be slow due to random read operations usually being slower than range scan operations in very large distributed databases, such as Bigtable and PNUTS [SSC<sup>+</sup>08] (developed by Google and Yahoo! respectively). These databases can be modeled as Distributed Ordered Tables (DOTs) that partition keys into regions to support range queries on the keys [CCF<sup>+</sup>08, CRS<sup>+</sup>08, SF09]. It is also commonplace to use non-key columns in ordered tables for secondary indexes, an example of which can be found in SeqWare [OMN10]. However, range queries over secondary indexes can be slow due to random reads being slower than scans [big08]. [ZLW<sup>+</sup>10] Clustering indices can reduce the number of random read operations, but requires more storage space where data is replicated.

Moreover, other NoSQL databases, such as MongoDB [Cho13], Couchbase [Bro12], Redis [Car13] and Memcached [Pet08], MemcacheDB [TB11] and ClusterPoint [RE13] have also become a popular solution for managing large volumes of data. MongoDB is a cross-platform document-oriented database, which avoids the traditional relational database structure in favor of JSON-like documents with dynamic schemas (BSON), making the integration of data in certain types of applications easier and faster.

### **2.2.1 HBase: An open source implementation of Google Bigtable**

HBase is an Apache Software Foundation open-source implementation of Bigtable written in the Java programming language. HBase is a sub-project of Apache Hadoop [Whi12], which has HDFS [Bor08] as the distributed file system and MapReduce as the parallel

computing model. There are three major components of HBase. The HBase Master assigns Regions to HRegionServers. HRegionServers handle client read and write requests. The HBase client discovers HRegionServers, which are serving a particular row range of interest. HBase builds on top of HDFS, has one or more HMaster and many HRegionServer to manage data regions, as shown in Figure 2.12. ZooKeeper [HKJR10] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

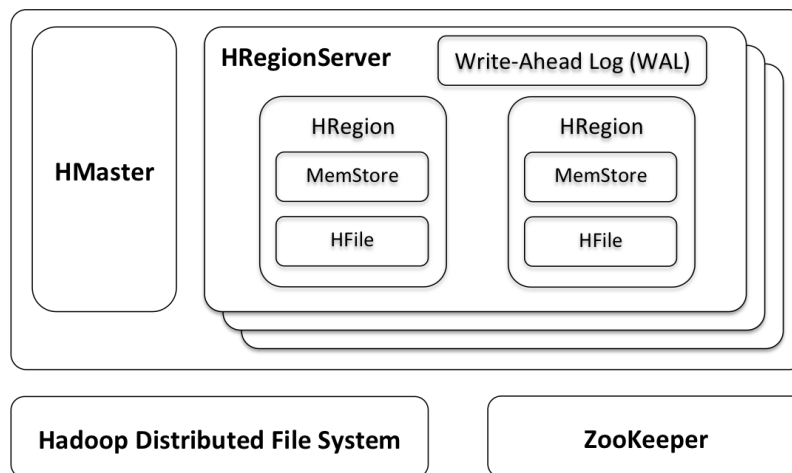


Figure 2.12: HBase architecture

To deeply understand the essence of the Key Value schema design, the data organization and query operation of HBase are introduced to fully describe HBase behaviour.

HBase stores most of data in disks using the HFile structure. HFile, based on Hadoop's TFile, stores data by mimicking the SSTable format used in Google's Bigtable architecture. These files are primarily handled by the HRegionServer. The HRegionServer divides HFiles into smaller blocks when stored within the Hadoop Distributed Filesystem (HDFS). Thus, the HDFS data copy number and data block size may influence the data query performance greatly.

All blocks in a HFile have a variable length except FileInfo and Trailer blocks. As show in Figure 2.13, the Trailer uses pointers to link to the other blocks and is written at the end of the file. The Index blocks record the offsets of the Data and Meta blocks. Both

the Data and the Meta blocks are optional. Key Value pairs are stored in Data blocks. The block size is configured by the HColumnDescriptor. The "Magic" header before Key Value pairs indicates whether these pairs are compressed or not.

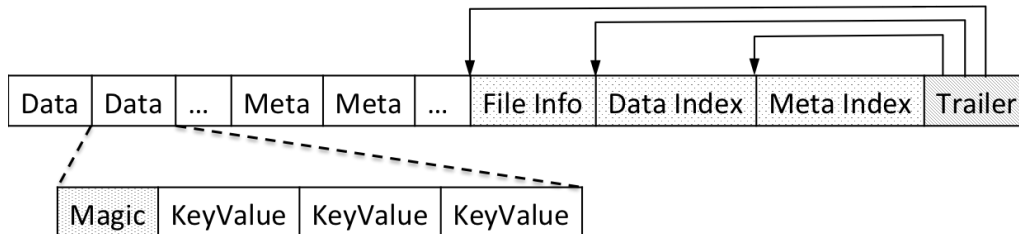


Figure 2.13: HFile structure

The default data block size is 65535 bytes. Larger block size is preferred if files are primarily for sequential access. However, it would lead to inefficient random access because there is more data to decompress. Smaller blocks are efficient for random access, but require more memory to load the block index, and slower to create due to more I/O flush operations. Furthermore, given the internal caching in Compression codec, the smallest possible block size would be around 20KB-30KB.

Essentially, each KeyValue in the Data block is simply a low-level byte array. Figure 2.14 shows the structure of the Key Value pairs. The first two fixed-length numbers indicate the size of the key and the value part. With that information, the value can be directly accessed by offsetting into the array without reading the key. The key contains Row (Row Key), Column Family (Family) and Column Qualifier (Qualifier), which could be used as composite key to speed up queries.

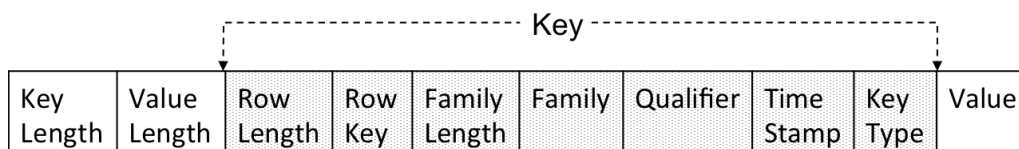


Figure 2.14: the structure of the Key Value pairs

When a query arrives, the query client first contacts the Zookeeper to find the location of the region that hosts a particular row key. Then the corresponding HRegionServer creates

a HRegion object for that region and sets up a Store instance for each HColumnFamily. An HRegion also has a MemStore and a HLog instance. "Write-Ahead-Log" (WAL) uses MemStore and a HLog to handle physical operations, such as read, write and cache. Each region name is encoded using a Jenkins Hash function and a directory is created for it. Thus, the full path structure shows as below:

```
/hbase/<tablename>/<encoded-regionname>/<column-family>/<filename>
```

For example, as shown in Figure 2.15, the Client first connected to ZooKeeper to find the location of the HRegionServer containing system table -ROOT- (Region Server I). Then the -ROOT- table returned the location of regions that store .META. table (Region Server II and III). The Client loaded all these meta information and then search for row-2 of TableA in the meta data. Finally, Region Server IV was connected to serve for this query.

So both advantage and disadvantage are easily understood based on the HBase architecture design. The advantage is HBase supporting the large sequential data query by the Row Key in one or a few Families. This range query operation can be performed in a very high speed. The disadvantage of HBase is slow queries by other attributes in Qualifiers. Based on the HBase data query method and data cache features, how to make use of the advantage and avoid the disadvantage becomes a real challenge in actual use cases. CCIndex shows a possible solution. [ZLW<sup>+</sup>10]

### 2.2.2 CCIndex

Distributed hash tables (DHTs) [SMK<sup>+</sup>01, RFH<sup>+</sup>01, RD01, ZKJ<sup>+</sup>01] are scalable and reliable for key-value pair storage. Because the data is partitioned by hashing functions, DHT systems do not support range queries naturally. MAAN [CFCS04] and SWORD [AOVP08] use locality preserving hashing and store attributes in DHT as index to support range queries. However, the logN hop latency is not good for user-interactive applications.

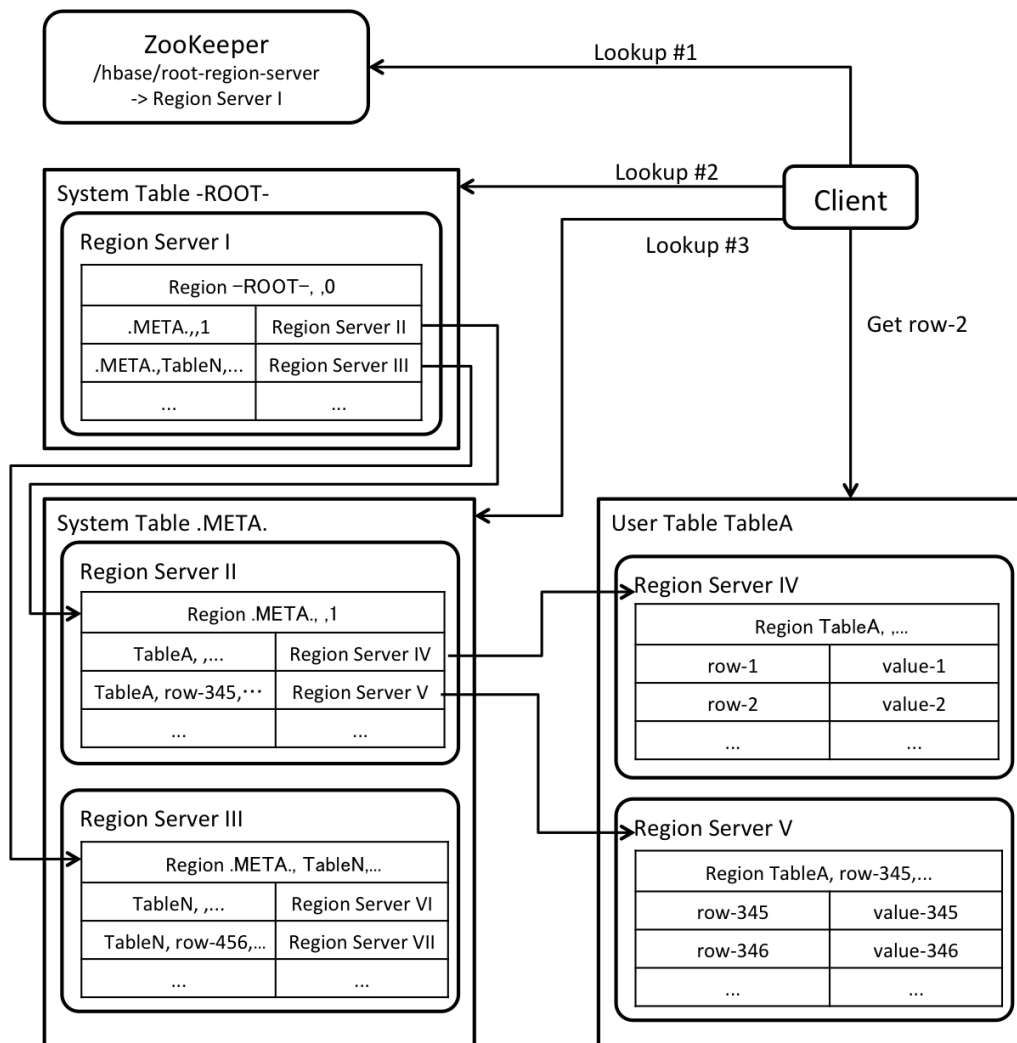


Figure 2.15: HBase query example

A solution to supporting high dimensional range queries over DOTs is CCIndex [ZLW<sup>+</sup>10]. CCIndex creates multiple complementary clustering index [BE77] tables, each with a search column containing a full row of data, allowing range queries [AL05] to act as range scans. CCIndex takes advantage of partition-to-server mapping [PG11] information to estimate the size of the results for each sub query. To reduce storage overheads, CCIndex disables the underlying data replication mechanisms and creates a check table for each search column to allow incremental data recovery.

CCIndex was originally designed for tables with 3 to 5 indices, heavy query operations and few updates. The first application, which CCIndex was applied on, was a system mon-

itor tool mimicking Nagios. This monitor tool collected millions of monitoring records and store in CCIndex. The data schema used in the tool is shown below in Figure 2.16. The [ZLW<sup>+</sup>10] shows CCIndex can provides more than 10 times faster query than secondary index implemented in HBase and 2 times of speed of MySQL Cluster.

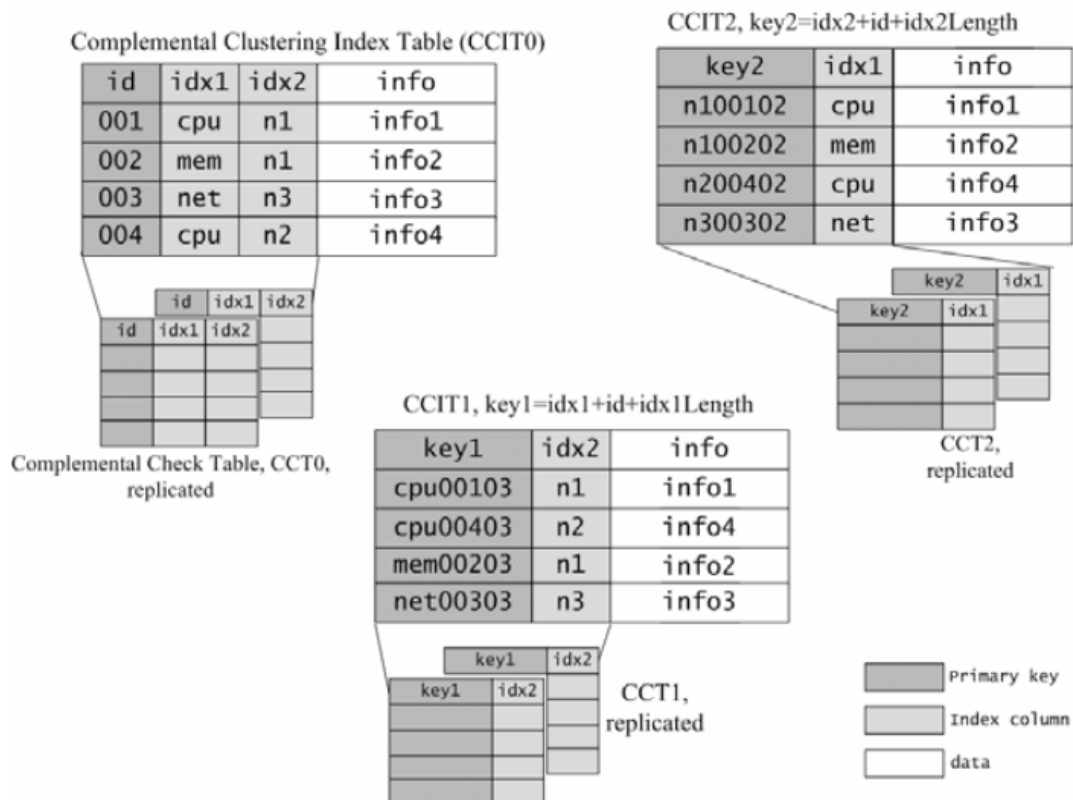


Figure 2.16: CCIndex architecture [ZLW<sup>+</sup>10]

To address the bottleneck of data queries on expression data in tranSMART, CCIndex has been built on to extend for use with HBase and Hadoop. Theoretical calculations find that CCIndex typically consumes up to 22% more storage space than using a traditional secondary index. However, CCIndex also demonstrates a 5.5 times increase in throughput of range queries than using secondary indexes. If a data model were properly designed based on the complementary clustering index, the high query throughput would speed up the data preparation in typical statistical workflows carried out in tranSMART.

### 2.2.3 MongoDB

MongoDB [Cho13] was first developed by the current software company MongoDB in October 2007 as a component of a planned platform as a service product and then shifted to an open source development model in 2009, with MongoDB Inc offering commercial support and other services. Since then, MongoDB has been adopted as backend software by a number of major websites and services, such as Craigslist, eBay, and the New York Times. MongoDB supports many features, such as full index, auto-sharding, etc.

Any attribute in MongoDB can be indexed using single field indexes or compound indexes. MongoDB also supports indexes of arrays, called multi-key indexes, as well as indexes on geospatial data.

Sharding is the process of storing data records across multiple machines on demand of data growth. In the big data age a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. As shown in Figure 2.17, the mongod in a shard stores actual data, the configuration servers store sharding configuration, the mongos runs on an application server as a stateless router.

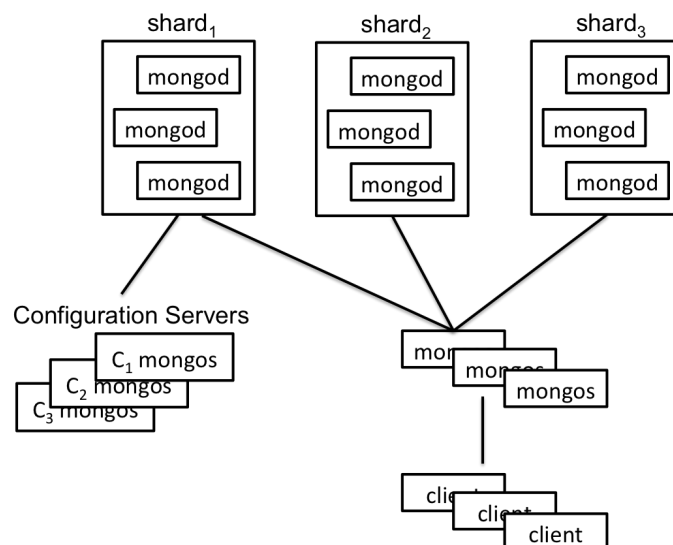


Figure 2.17: The architecture of MongoDB sharding. [Cho13]



In this thesis MongoDB is mostly used to implement relational data model to test how relational models perform in Key Value databases.

# Chapter 3

## Generic -omics data model

### 3.1 Introduction

Information from genomic, proteomic and metabolic measurements has already benefited identification of disease subgroups and the prediction of responses of individual subjects, which is known as molecular profiling based patient stratification [SF05]. Biomedical research is moving towards using high-throughput molecular profiling data to improve clinical decision making.

A typical molecular profiling database contains values of features from individuals of interest or samples, to be used for several medical studies. A subject of study is usually a particular sample to analyze for the specific study. At high level, queries on bioinformatics databases are meant to retrieve subjects based on their features as searching conditions. Thus, typical molecular profiling data analysis contains the following three classes of queries:

The first step is the marker selection that chooses proper features for further analysis, such as classification or clustering. In this step, multiple cohorts with all markers in the

database are collected for a particular study, for example "asthma".

```
select * from table
where subjects in {the cohorts}
and study = 'asthma';
--Query 1
```

If the features of a study are known already, the particular features of all the samples in the study are selected for the analysis. For example, users may want to retrieve the samples genotyped at a particular DNA position in the "asthma" study.

```
select * from table
where markers in {the DNA positions}
and study = 'asthma';
--Query 2
```

If a relationship between features and diseases are found, all the samples in the database are selected to detect potential patients for a particular disease. For instance, users may need all the samples genotyped at a particular DNA position.

```
select * from table
where markers in {the DNA position};
--Query 3
```

Many management systems can enable scientists to do that, such as tranSMART [ABHG13], NCBI [BWL<sup>+</sup>13] and SeqWare [OMN10]. The storage of the first two platforms use SQL model and the last one uses a Key Value model. TranSMART is a biomedical data warehouse and analytics software platform that integrates clinical and genomics data using SQL models. The NCBI dbSNP [SWK<sup>+</sup>01] individual genotype data schema is a widely

used relational data model for SNP data. The SeqWare implements a Key Value model based on HBase with secondary indices.

In the big data era, molecular profiling data size increases sharply due to new biological techniques, such as next generation sequencing. None of the existing databases work well whilst considering the three "V" features of big data (Volume, Variety, and Velocity). An alternative solution is to take advantage of emerging NoSQL techniques with high speed indices to speed up queries over molecular profiling data.

Especially, the range scan query in Bigtable can perform even faster than indices of other NoSQL databases for molecular profiling data. For example, [WPW<sup>+</sup>14] indicates HBase data query performs up to 7x faster than MongoDB for microarray data. Yet random read operations in Bigtable can be extremely slow due to the large size of the Bigtable data block. Thus, in Bigtable the secondary index [LY01] has a smaller space requirement but performs slow; the clustering index [ZLW<sup>+</sup>10] has larger storage overhead but performs fast. The global index has high network traffic but is easy to implement; the local index has low network traffic but is difficult to implement. [FYL<sup>+</sup>15] did a performance test on all those indices. The throughput of global clustering index is best with a reasonable storage overhead.

However, none of existing molecular profiling data models work well for Bigtable when considering the three "V" features. To improve the throughput of the current models, this chapter introduces a collaborative generic data model (CGDM), which uses three optimized complementary global clustering index tables (CGCITs) for each of the queries described above. Each CGCIT uses vertical partitions to further speed up the queries and increase the data availability. Thus, CGDM can solve the velocity and variety issues with limited extra volume cost.

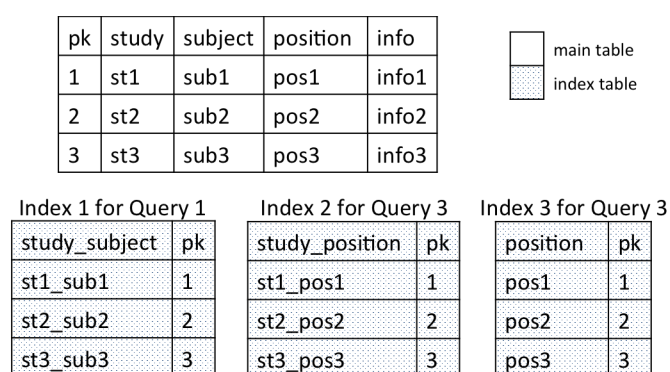


Figure 3.1: Traditional molecular profiling data model

## 3.2 Methods

A typical traditional molecular profiling data model (TDM) used in tranSMART includes a main table and three secondary index tables, as shown in Figure 3.1. The pk is a unique id. Main table stores data ordered by the id. Three secondary indices are created to speed up the three typical queries above. The subject id is usually associated with its study, so study and subject together can be a unique id for Query 1. The combination of study and DNA position is the fastest index to locate a DNA position in a specific study for Query 2. The position alone can be used to search information on a specific DNA position in the database for Query 3.

There are two types of secondary indices that can be used for this purpose. One stores pk values in the main table, while the pk column in the other index stores pointers linking to the physical location in the main table. In a distributed database, it is very difficult to maintain the second type of index. For example, if the distributed table blocks are merged or split, all the related physical locations in the pk column will be re-calculated. Thus, the pk in the index table usually contains the pk value in the main table. That means if you sequentially search the index table to find the pk, you need to randomly search the pk in the main table to find the data. The random search leads to a big challenge to design a multi-dimension index for large range query in Bigtable.

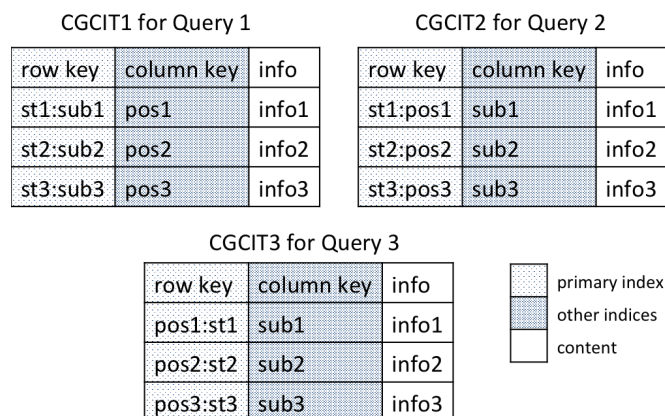


Figure 3.2: CGDM structure.

A clustering index may be introduced to solve the random search problem from the secondary index. But the clustering index leads to a big storage overhead, as each index table is a full copy of the main table. In order to reduce the storage overhead, the backup copies in the database can be re-used. Fortunately, in Bigtable there is usually three copies of each table to make sure the data consistency and fault toleration. A main table with two index tables does not increase any overhead. Furthermore, if the main table can be even re-used, three collaborating clustering index tables can be created for the 3 typical queries.

The remaining work is how to design the index for each query. From the analysis view, the study, subject and DNA position are frequently used as search conditions; from the database view, Bigtable offers a hierarchical composite primary key, including row key (primary index) and column key (other indices). Bigtable horizontally partitions a table by row key and vertically partitions the table by column key. The row key is the primary index of the key, so the main index columns in this index table are stored here. The other index columns can be added in the column key. Based on the principles above, a collaborative data model for molecular profiling data is created, as shown in Figure 3.2. In addition, CGCIT3 cannot follow Index3 in Figure 3.1 in case that, with study number increases, the size of data in one DNA position exceed the database limit.

Moreover, except the usage of horizontal index above, CGDM also consider utilizing

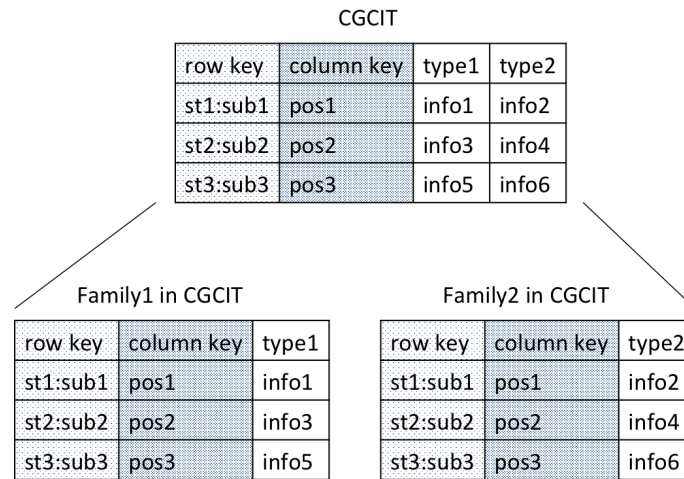


Figure 3.3: Structure of multiple types of data.

the Bigtable vertical partition feature, Family, to further speed up data query, as shown in Figure 3.3. Family is a special concept, related to a group of columns that contains the same type of data. The same type of data are usually stored together and retrieved together. For example, three different types of gene expression data may be stored, such as raw value, logarithm value and zscore value, and a normal query only focuses on one type of data. If three Families are used for the three types of data, the query could be sped up by searching about 1/3 of the data only.

### 3.2.1 Fault Tolerance

In CGDM, CGCITs have no replica to avoid huge storage overhead, and cause the problem of fault tolerance. The basic idea is that CGCITs replicate and recovery each other in record level. If there are multiple Families in the CGCIT, the damaged block can find all row keys by checking another Family and then find the data in another CGCIT. If only one Family exists, a CGCIT can be recovered by another CGCIT. As both CGCIT1 and CGCIT2 row keys begin with study column, rows in CGCIT1 and CGCIT2 are both ordered by study. When a region of a CGCIT1 is damaged, users can quarantine the regions of the studies involved and use regions containing the same studies in CGCIT2 to

reconstruct the regions. For CGCIT2, the situation is similar. For CGCIT3, users can isolate DNA positions involved and fetch corresponding information by searching the whole CGCIT2. During the scan, the DNA position range can skip unnecessary regions.

### 3.3 Generic data model for molecular profiling data

Based on the methods in the previous section, a new data model is created, as shown in the following Figure 3.4. This model is only a basic template, so some adjustments may be applied for specific data types. In addition, to make the table name easy to remember, CGCIT1 is also called the Subject table. So are CGCIT2 (Position) and CGCIT3 (Cross-study).

#### 3.3.1 Subject table

In the subject table the key is a composite key including fixed length StudyID and SubjectID, while the value consists of several families of different data types, such as annotation data and expression values.

#### Determining Row Key and Column Key of transcriptomic data

In order to speed up queries, the StudyID and SubjectID are placed in the Row Key, as shown in Table 3.1 . This is for two reasons. The first reason is that a Family can manage a subject of data efficiently. When a Family is used in this way, all attributes larger than the subject (StudyID and SubjectID) are placed in the Row Key and attributes smaller than the Family (PositionID) are placed in the Qualifier. The second reason is that all records belonging to a Family are stored together in several HFiles. As shown in Figure 3.4, only data belonging to Family A are stored in Region I and data in Family C are stored in



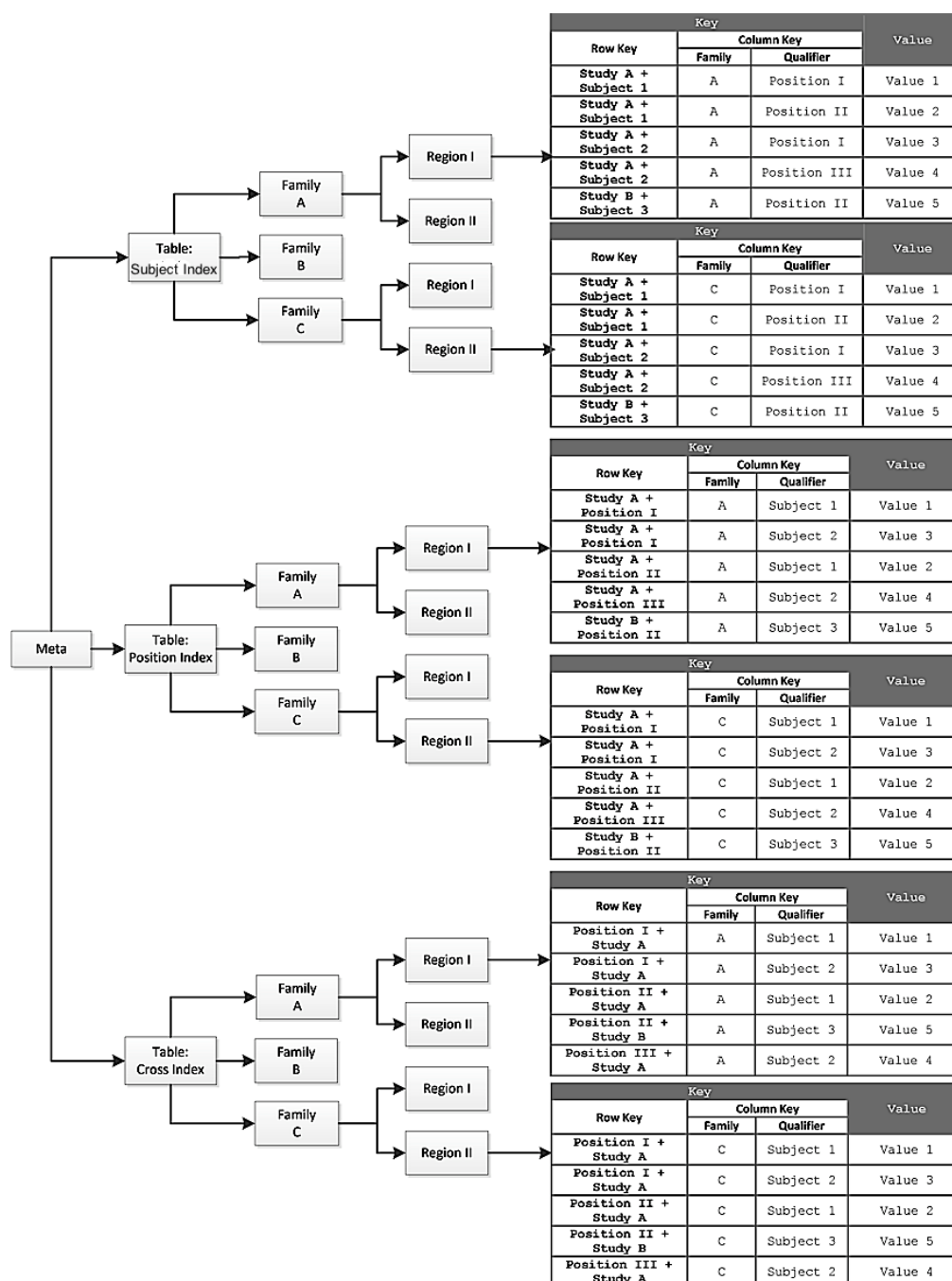


Figure 3.4: Generic data model. StudyID is the identity of clinical trial or the name of a dataset, SubjectID is the identity of a subject in an assay, PositionID is a gene or protein location expressed in an assay and Others is additional information.

Region II. For a typical use case when multiple identical subjects are retrieved, the HFiles storing the data in these subjects will be loaded into caches. Therefore, only retrieving the first record causes a significant delay when loading data from disk to memory, while

all other records in the same HFile are fetched directly from memory caches. This design takes full advantage of caches.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Study A + Subject 1	A	Position I	Value 1
Study A + Subject 1	A	Position II	Value 2
Study A + Subject 2	A	Position I	Value 3
Study A + Subject 2	A	Position III	Value 4
Study B + Subject 3	A	Position II	Value 5

Table 3.1: Subject table

### Optimizing the Row Key to speed up data location

The order of the two fields in the Row Key also has a very significant effect on the query performance. Typical user cases always begin with a gene information query of several subjects based their cohorts or clinical measurement results. By placing the StudyID before the SubjectID (StudyID + SubjectID), the composite key becomes two indices on the study and subject respectively. As shown in Table 3.1, Study A appears before Study B and within Study A, Subject 1 appears ahead of Subject 2. When multiple identical subjects in a study are retrieved, StudyID + SubjectID can precisely locate all HFiles expected by using StudyID + SubjectID to compare to the start keys of an HFile.

### Optimizing the Column Key to increase cache hit rate

One way to further increase cache hit rate is to design a new structure by classifying different types of data under Column Key. The Family divides different types of data (info and subject) into different HFiles and the Qualifier orders positions by PositionID lexicographically. As shown in Table 3.1, Qualifier Position I always appears ahead of Position II or Position III. In the motivating example above, the user requests data on

several subjects that contain only expression values, but the query returns millions of records. Such large numbers of expression records can be loaded with only a limited number of disk loading operations through the corresponding Column Keys.

This Subject table only provides fast queries by subject within one study. However, if queries over an genetic position or positions across studies are required, these queries may not be performed as fast as the queries over subjects. In order to provide faster query by position as well, a supplementary cluster index is created.

### 3.3.2 Position table

In the Position table, as shown in Table 3.2, StudyID and PositionID are composed of the key. This table offers high-speed query by position within a study.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Study A + Position I	A	Subject 1	Value 1
Study A + Position I	A	Subject 2	Value 3
Study A + Position II	A	Subject 1	Value 2
Study A + Position III	A	Subject 2	Value 4
Study B + Position II	A	Subject 3	Value 5

Table 3.2: Position table

#### Determining Row Key and Column Key of transcriptomic data

In order to accelerate queries by genetic position, the StudyID and PositionID are placed in the Row Key. Thus, the SubjectID is moved to Qualifier, as shown in Table 3.2. All subjects in the same position are stored together in HFiles. For a typical use case when one or multiple identical positions in a specified study are retrieved, the HFiles storing the data of these positions will be loaded into caches to take advantage of these caches. Therefore, only retrieving the first record causes a significant delay when loading data

from disk to memory, while all other records in the same HFile are fetched directly from memory caches.

### **Optimizing the Row Key to speed up data location**

The order of the two fields in the Row Key affects the query performance greatly. Typical user cases may start with a gene information query of several genes or probe sets in a study. By placing StudyID ahead of PositionID, the composite key becomes two indices on the study and genetic information respectively. As shown in Table 3.2, Study A is always placed before Study B. All Positions in Study A are ordered by positionID (I, II, III). When multiple identical genetic attributes are retrieved, this Row Key can precisely locate all HFiles expected by using this Row Key to compare to the start keys of a HFile.

### **Optimizing the Column Key to increase cache hit rate**

The Family divides different types of data (A, B, and C) into different HFiles and the Qualifier (SubjectID) orders subjects by SubjectID. As shown in Table 3.2, Subject 1 always appears before Subject 2 before Subject 3. If queries by genetic position are required, the required data on each position are stored together. Such large numbers of expression records can be loaded with only a few disk reading operations through the corresponding Column Keys.

### **3.3.3 Cross-study index table**

The key of position index table consists of PositionID and StudyID, as shown in Table 3.3. The Column Key structure is the same as the Position table. This table is utilized to query by position across studies.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Position I + Study A	A	Subject 1	Value 1
Position I + Study A	A	Subject 2	Value 3
Position II + Study A	A	Subject 1	Value 2
Position II + Study B	A	Subject 3	Value 5
Position III + Study A	A	Subject 2	Value 4

Table 3.3: Cross-study table

### Optimizing the Row Key to speed up data location

The order of the two fields in the Row Key has different function from the Position table. Some user cases may start with a gene information query of several genes or probe sets across studies. By placing PositionID ahead of StudyID, the composite key becomes two indices on genetic position and the study name respectively. As shown in Table 3.3, Position I is always placed before Position II and III. Study A and B in Position II are ordered by StudyID, A ahead of B. Thus, all expression data in Position I are stored together. When one or more identical genetic positions are retrieved, this Row Key can precisely locate all HFiles expected by using this Row Key to compare to the start keys of an HFile. All these expression records can be loaded with only a few disk operations through the corresponding Keys.

## 3.4 Performance evaluation methods

CGDM is aimed at solving the velocity issue for molecular profiling data, so improving the throughput of each query described in the introduction is the most important target. The method of benchmark design is to gradually increase the query size from a small value until the query throughput of the database becomes stable. There are two ways to increase the query size, including modifying the number of records in a query and

multiplying the queries with multiple threads.

Query 1 is usually a large query due to the high dimension of molecular data. For example, a microarray experiment could generate thousands of probe values for a subject, while a raw DNA sequencing experiment could generate millions of variants for a subject. A single thread Query 1 could fully occupy almost all resources in a database system and show the throughput of the system. The subject number for each query varies from about 100 to a few hundreds. For query Query 2 or 3, a single thread test cannot indicate the database throughput due the small query data size. Thus, multi-thread queries are introduced to test throughput. The thread number varies from 10 to 100.

In addition, if use cases with a medical meaning are given, the result will be more persuasive; if the query throughput is still increasing during the last few tests, larger queries will be added to fully evaluate the throughput.

### **3.5 Conclusion**

With a traditional molecular profiling data model using secondary indices, tranSMART faces the velocity issue. New NoSQL techniques, especially Bigtable, can solve the issue by improving the query throughput, but tranSMART's current data model does not work well for Bigtable. A new generic -omics data model CGDM, using the optimized complementary index and vertical partition, is designed to take advantage of Bigtable's high throughput query. Following CGDM principles, a detailed Key Value data model is created and explained. Finally, evaluation methods for query throughput are described as principles to evaluate implementations in the following chapters.

# Chapter 4

## Microarray Key Value Model

### 4.1 Introduction

A microarray is a two dimensional array on a solid substrate, which is usually a glass slide or silicon thin-film cell. It assays large quantities of biological material using high-throughput screening miniaturized, multiplexed and parallel processing and detection methods. The concept and methodology of microarrays was first introduced and illustrated in antibody microarrays by Tse Wen Chang in 1983 [Cha83] in a scientific publication and a series of patents. The "gene chip" industry began to grow significantly after the 1995 Science Paper [SSDB95] by Stanford University. With the development of companies, such as Affymetrix, Agilent, Applied Microarrays, Arrayit, Illumina, etc., the technology of DNA microarrays has become the most sophisticated and the most widely used. At the same time the use of protein, peptide and carbohydrate microarrays are expanding.

A DNA microarray is a collection of microscopic DNA spots attached to a solid surface. Researchers use DNA microarrays to measure the expression levels of large numbers of genes simultaneously or to genotype multiple regions of a genome. Each DNA spot

contains a specific DNA sequence, known as probes [HJCT88]. These can be a short section of a gene or other DNA element that are used to hybridize a cDNA or cRNA sample under high-stringency conditions. Probe-target hybridization is usually detected and quantified by detection of labeled targets to determine relative abundance of nucleic acid sequences in the target.

Since an array can contain tens of thousands of probes, a microarray experiment can accomplish many genetic tests in parallel. Therefore, arrays have dramatically accelerated many types of investigation. DNA arrays are different from other types of microarray only in that they either measure DNA or use DNA as part of their detection system.

The key principle behind microarrays [Qua01] is hybridization between two DNA strands, the property of complementary nucleic acid sequences to specifically pair with each other by forming hydrogen bonds between complementary nucleotide base pairs. A high number of complementary base pairs in a nucleotide sequence means tighter non-covalent bonding between the two strands. After washing off non-specific bonding sequences, only strongly paired strands will remain hybridized. Total strength of the signal from a spot depends upon the amount of target sample binding to the probes present on that spot. Microarrays use relative quantitation in which the intensity of a feature is compared to the intensity of the same feature under a different condition, and the identity of the feature is known by its position.

Due to the biological complexity of gene expression, the considerations of experimental design that are discussed in the expression profiling article are of critical importance if statistically and biologically valid conclusions are to be drawn from the data.

Microarray data sets are usually very large, and analytical precision is influenced by a number of variables. Statistical challenges include taking into account effects of background noise and appropriate normalization of the data. Normalization methods may be suited to specific platforms and in the case of commercial platforms the analysis may be



proprietary.

An increasing amount of biological data being produced has predicated the use of databases capable of storing and analyzing genomic expression data. In the field of translational research, knowledge management platforms use databases to store a variety of data produced from clinical studies, including patient information, clinical outcomes as well as high-dimensional -omics data. For optimal analyzes and meaningful interpretations, such databases also store legacy data taken from public sources, such as the GEO and the Gene Expression Atlas, alongside new study data. This enables cross-study comparisons and cross-validation [BFT<sup>+</sup>05] to take place. High-throughput transcriptomic data generated by microarray experiments is the most abundant and frequently stored data type currently used in translational studies.

A frequently used query is fetching patient gene/protein probe set expression levels in a study in order to perform data analysis. For each study, the transcriptomic data stored in the database is comprised of individual probe set values for each patient sample (in some cases multiple samples from each patient are profiled), and annotation information further describing the experiment (e, g. trial information, microarray platform identifier, normalization method, gene descriptions). Once retrieved the data is passed to analytical tools, such as GenePattern [RLG<sup>+</sup>06] or custom R workflows for further analysis or visualization.

### 4.1.1 Classic relational data model - I2B2

The data mart Data Repository Cell (also named the Clinical Research Chart, or CRC) of Informatics for Integrating Biology and the Bedside (I2B2) is a widely used relational data model for high dimensional data [BBK01], which is originally designed to store data from different types of clinical data such as clinical trials, medical record systems, laboratory systems, and many other types of clinical data. Given the complexity of the full I2B2

database model, the main data tables are abstracted in order to explain the I2B2 model more clearly. The CRC stores this data mainly in two tables, the patient and observation tables, as shown in Figure 4.1. In addition to these two tables, there are two lookup tables, the concept and provider tables. The two data tables, along with two of the lookup tables make up the star model of the warehouse, with the central fact table surrounded radially by one or more dimension tables. The most important concept regarding the construction of a star model is identifying what constitutes a fact.

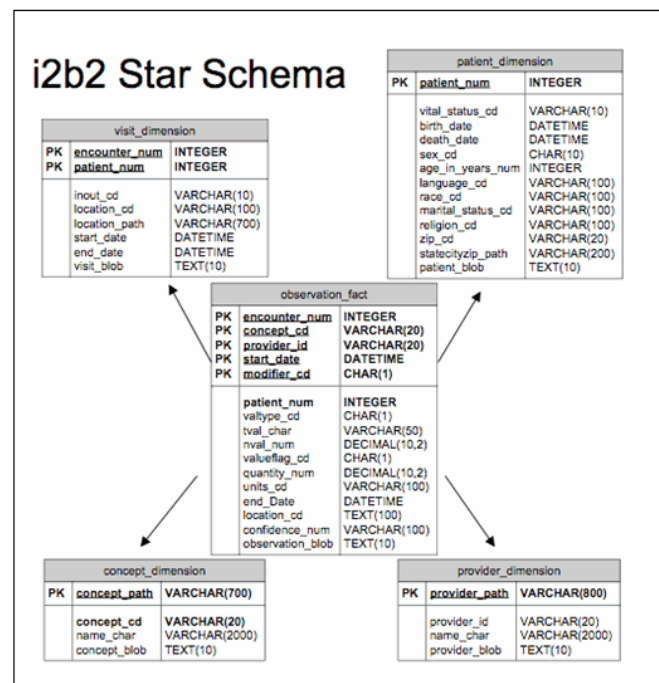
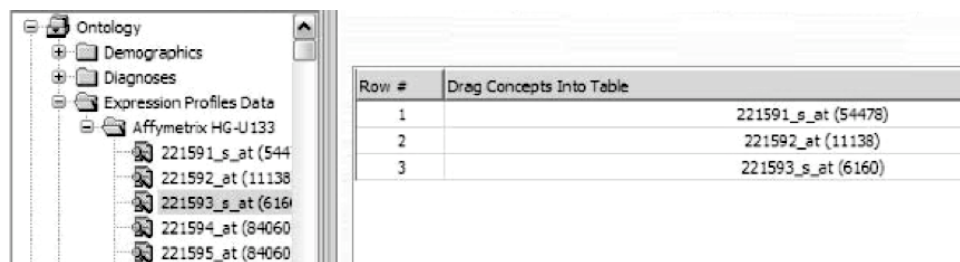


Figure 4.1: I2B2 data model [MWM<sup>+</sup>10]

The fact table contains the observation value, such as clinical measurements and gene expression levels, and basic attributes about the observation, such as the patient and provider numbers, a concept code for the concept observed. In I2B2, the fact table is called `observation_fact`. The `observation_fact` table is the central table of the I2B2 star model and represents the intersection of the dimension tables. Each row describes one observation about a patient. Most queries in the I2B2 database require joining the `observation_fact` table with one or more dimension tables together, such as `patient_dimension` or `concept_dimension`.

Dimension tables contain information about the data organization, such as data categories and summaries, which are adopted for further descriptive and analytical information about attributes in the fact table. In the I2B2 Data Mart, there are four dimension tables providing additional information about fields in the fact table: `patient_dimension`, `concept_dimension`, `provider_dimension` and `visit_dimension`. Each record in the `patient_dimension` table represents a patient in the database. The patient information contains demographics fields such as gender, age and race. The `concept_dimension` table contains one row for each concept. Possible concept types are diagnoses, procedures and lab tests. The structure of the table is significantly flexible to support most concept types, such as demographics and genetics data. The `concept_path` illustrates the concept's hierarchy. The `concept_cd` represents the code for diagnosis, procedure, or lab tests. The `provider_dimension` table contains a note from the provider. The `visit_encounter_number` in `visit_dimension` is the unique id for each observation. Other information about the visit, such as hospital locations, is also stored in `visit_dimension` table.

For example, in Figure 4.2 a gene expression dataset from Affymetrix HG-U133 chip [LLS<sup>+</sup>03] is stored in a I2B2 model. The `concept_dimension` table contains each probe sets a row, such as `221591_s_at` and `221592_at`. The corresponding patient and provider information are stored in `patient_dimension` and `provider_dimension` tables respectively. The `observation_fact` table contains each expression value and its basic attributes per row.



Row #	Drag Concepts Into Table
1	221591_s_at (54478)
2	221592_at (11138)
3	221593_s_at (6160)

Figure 4.2: I2B2 example

### 4.1.2 TranSMART high dimensional data model

Inspired from I2B2 data model, tranSMART inherited the star model and improve the fact table by providing more flexible data types, such as high dimensional type. The high dimensional data type is more suitable for gene expression data than the basic numeric type. A row in this data type links to a partition in another table named microarray that contains gene expression values and basic attributes, such as SUBJECT\_ID and PROBESET\_ID. Figure 4.3 shows a snapshot of tranSMART performing high dimensional data analysis.

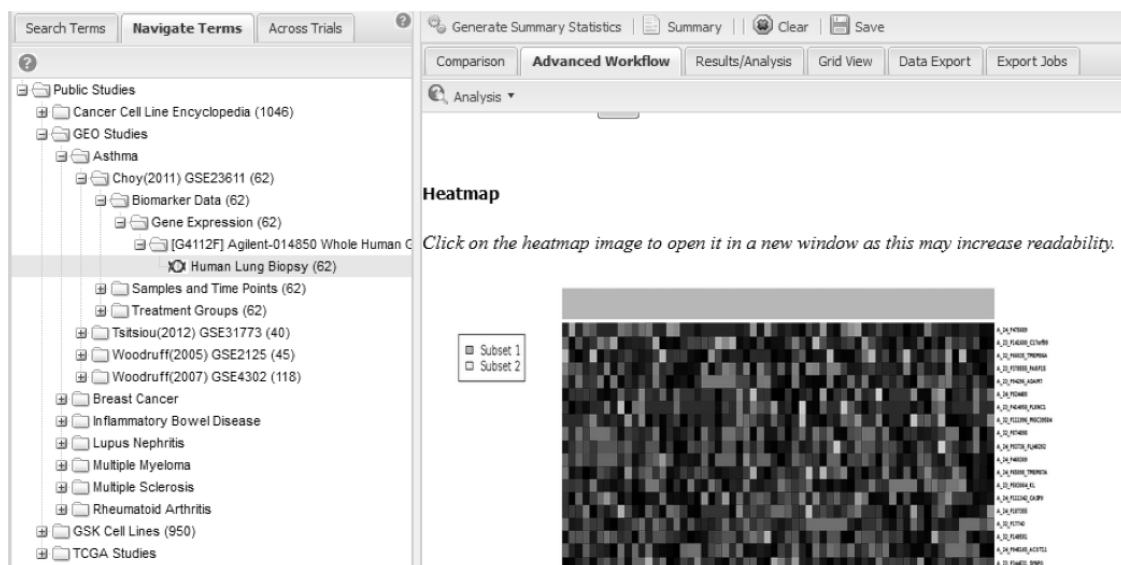


Figure 4.3: tranSMART example

Table 4.1 shows a typical structure of a microarray table that is used in a relational database system. This example is based on the DEAPP model used in the implementation of tranSMART 1.1. PROBESET\_ID, SUBJECT\_ID, STUDY\_NAME correspond to basic annotation data; RAW, LOG and ZSCORE are probe set values [CVFB03].

All records in the table are organized by a B-tree like structure, such as B-tree, B+tree or B\*tree. B-tree structures are balanced search trees that are optimized for large amounts of data. These structures allow search, access, and deletions of tree nodes in logarithmic time. B-tree like structures are commonly used in the implementation of databases and file systems and are fully described in [BM72]. The time to fetch the patient probe set

PROBESET_ID	VARCHAR2(100)
SUBJECT_ID	NUMBER(18)
STUDY_NAME	VARCHAR2(50)
RAW	NUMBER(18)
LOG	NUMBER(18)
ZSCORE	NUMBER(18)

Table 4.1: TranSMART microarray data model. PROBESET\_ID represents a genetic identity, SUBJECT\_ID is a subject identity, STUDY\_NAME shows a study identity, and RAW, LOG and ZSCORE are three types of expression values.

data using a relational model, such as that in Table 4.1, is therefore:

$$\sum t_r = p_n * p_s * \log_m(p_n * p_s) * t_r \quad (4.1)$$

where the time to find a next node in the tree is one standard unit, the order of the tree is  $m$ , the time to fetch a record in relational database is  $t_r$ , number of patients is  $p_n$ , and the number of probe sets for each patient is  $p_s$ .

What this illustrates is that in order to traverse the tree to retrieve a large study, a significant number of records need to be read from the physical disk where such a large number of database operations will cumulatively be slow. A common solution for scalability issues in relational databases is database partitioning [ECS<sup>+</sup>08], which allows for a logical database to be divided into constituent parts and distributed over a number of nodes, for example in a compute cluster. This approach has enabled relational databases to support large-scale genomics datasets through horizontal partitioning (also referred to as sharding), however, this still does not solve the data retrieval performance issues such as those observed in translational research studies loaded into tranSMART.

An alternative solution, and what forms the contribution described in this chapter, is to utilize a NoSQL database modality, the Key Value pair data model, as implemented in NoSQL databases such as Google BigTable. The first advantage of such a data model is that it typically maintains data in lexicographic order by Row Key. The second advan-

tage is that the Column Key includes two parts: a Family and a Qualifier, where database columns (Qualifiers) are grouped into sets (Families) and all data stored in a Family is usually of the same type and is compressed and stored together. When a key is retrieved, a Key Value array is loaded into memory and the expected Key Value pairs are returned. Taking these features of the Key Value model into account, I hypothesized that a Key Value pair data model would be more performant in data retrieval than the current relational model used for microarray data storage in tranSMART. This chapter describes an experiment using a new database model for one of tranSMART's microarray data tables that may be more suitable for high-dimensional data storage and querying than the relational model currently implemented in tranSMART.

The current version of tranSMART operates with SQL-based databases in a single node mode, typically as single PostgreSQL [Mom01] 9.3 or Oracle 11g [BDDY09] database. It is possible to migrate the current tranSMART data model to SQL-based database clusters to solve the performance problem mentioned in motivating example. For example, MySQL Cluster [RT04] is an open-source high performance database cluster. MySQL Cluster consists of multiple network database nodes (NDB), a single management node (MGM) and multiple MySQL nodes. The data is horizontally partitioned across the NDBs, where each MySQL node is responsible for handling SQL operations. The MGM controls all NDB and MySQL nodes. MySQL Cluster can use the same relational data model detailed previously in Table 4.1.

NoSQL document storage systems, such as MongoDB [Cho13], CouchDB [ALS10], Riak [Cat11] are popular alternatives to using relational database management systems such as MySQL, because relational data models can easily map to a document-based Key Value model. For example, MongoDB has features such as indexing in the form of B-trees, auto-sharding and asynchronous replication of data between servers. MongoDB stores data in collections and each collection contains documents. Each document is a serialized JSON [Cro06] object. MongoDB used relational model where each column is represented

by a field in a JSON object. For example, the relational data record in Table 4.2 maps to the JSON object shown in Figure 4.4. Note that the field `_id` holds a unique key generated for each JSON object.

PROBESET_ID	SUBJECT_ID	STUDY_NAME	RAW	LOG	ZSCORE
204454_at	79622	MULTMYEL	71.900002	6.16791991	8.3069731

Table 4.2: Example of a relational model representation of a patient record

```
{
  "_id" : ObjectId("51d997726a18db87555166a5"),
  "TRIAL_NAME" : "MULTMYEL",
  "PATIENT_ID" : 79622,
  "PROBESET_ID" : "204454_at",
  "RAW" : 71.900002,
  "LOG" : 6.16791991,
  "ZSCORE" : 8.3069731
}
```

Figure 4.4: Example of a JSON object that maps to the patient record illustrated in Table 4.2. The `”_id”` is automatically generated by MongoDB to distribute JSON data.

All of the data is imported into a single collection with the index built on the keys, where the sharding key uses `SUBJECT_ID` to distribute the data into multiple ranges based on `SUBJECT_ID`. NoSQL systems for structured data, such as BigTable and Cassandra [LM10], have been developed for managing very large amounts of data spread out across many servers. In contrast to document Key Value storage, such as MongoDB, NoSQL structured data systems do not easily map from a relational data model to a Key Value model. HBase is a popular example of such a system where it is part of the Hadoop framework that has been widely adopted for large-scale storage and processing. The Key Value microarray data model implementation using HBase is described in the following section.

## 4.2 Key Value Microarray Model

As show in Figure 4.5, following the generic model, there are three tables in this schema and each of them uses a different index for a typical expression data query. The Subject table is designed to find significant differential gene or protein expression data. With clinical information in a trial, such as cohorts and clinical measurement statistics, all available differential expressed genes of the specified subjects with typical features are located for further analysis to find the expected genes. After differential expressed genes are calculated, Position table will be used to find the top hundred genes in several other studies to further reduce the gene number. If some genes associated with certain traits or diseases are found, queries over Cross-study table are generated to detect whether a patient is vulnerable to certain diseases based on his expression levels of these genes.

In each table, data are split and compressed by Family to achieve higher query speed. A Family usually represents for a special data type, such as RAW, LOG, ZSCORE in microarray data. Different types of data are used by different functions. Normalized data are usually used more frequently than RAW data. Most of the functions only use one type of data. For example, marker selection analysis may load ZSCORE to find differential expressed probe sets and hierarchical clustering may be applied on LOG data. Thus, in marker selection, Family ZSCORE is searched while RAW and LOG data are not loaded to memory; in hierarchical clustering, only Family LOG is loaded for further computation.

### 4.2.1 Subject table

As shown in Table 4.3, the STUDY\_NAME and SUBJECT\_ID in Table 4.2 are placed in the Row Key, the data type (RAW, LOG or ZSCORE) is placed in the Family, the PROBESET\_ID is placed in the Qualifier and the expression value of a type (RAW, LOG or ZSCORE) is placed in the Value.



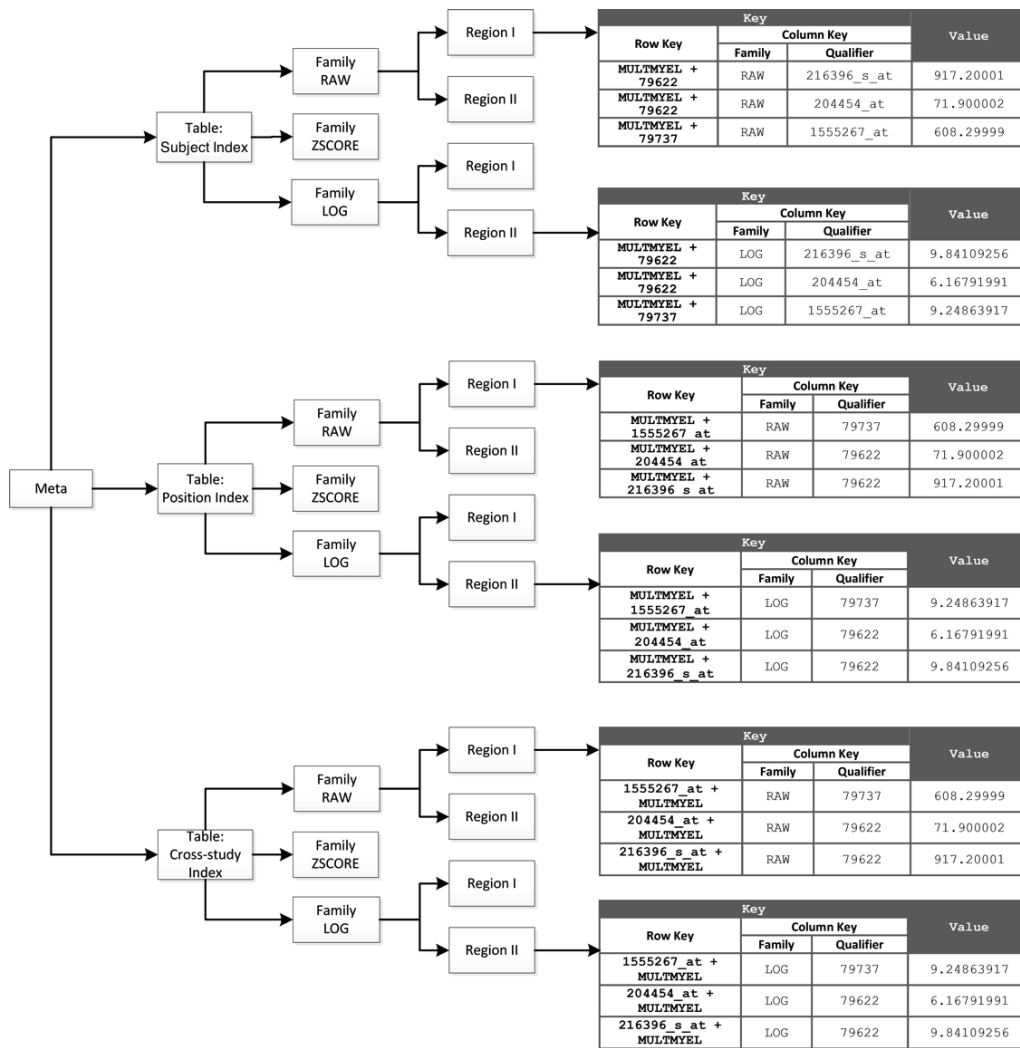


Figure 4.5: Microarray Key Value model

Key			Value
Row Key	Column Key		
	Family	Qualifier	
STUDY_NAME + SUBJECT_ID	DATA_TYPE	PROBESSET_ID	EXPRESSION_LEVEL
MULTMYEL + 79622	RAW	204454_at	71.900002
MULTMYEL + 79622	LOG	204454_at	6.16791991
MULTMYEL + 79622	ZSCORE	204454_at	8.3069731

Table 4.3: Key Value Microarray Subject Table schema.

The STUDY\_NAME and SUBJECT\_ID are placed in the Row Key for two reasons. For one thing, a Family can manage a single data type (RAW, LOG, or ZSCORE) of data efficiently. Then, all attributes larger than the data type (STUDY\_NAME and SUBJECT\_ID)

are stored in the Row Key and attributes (PROBESET\_ID) smaller than the data type are stored in the Qualifier. For another, all records belonging to a patient are stored together in several large HFiles. When multiple identical SUBJECT\_IDs are required, the HFiles can be loaded into caches. So, only fetching the first record leads to a long time delay due to the data loading from disk to memory, while the other records in the same HFile are read directly from memory.

Also, the order of the two fields in the Row Key is also very important. The composite key (STUDY\_NAME + SUBJECT\_ID) is a composite index on the study and patient. When subjects are searched, the Row Key can locate all HFiles by comparing the Row Key to the Start keys and End keys of a HFile.

Furthermore, Family divides different types of data (RAW, LOG, and ZSCORE) into different HFiles. A user usually requests a large amount of data but only one value type. Other types of data, which are not required, will not be retrieved. Thus, Family reduces the data searching and loading time.

### Key Value data model example

To better explain this design, its application is illustrated using the relational example in Table 4.4 and transforming it into the Key Value model shown in Table 4.5, 4.6, 4.7. These three tables show how a HFile stores these Key Value data on physical disks. Key Value pairs are ordered alphabetically by the Key in Table 4.4 and each pair is an array of Java data type byte.

PROBESET_ID	SUBJECT_ID	STUDY_NAME	RAW	LOG	ZSCORE
204454_at	79622	MULTMYEL	71.900002	6.16791991	8.3069731
216396_s_at	79622	MULTMYEL	917.20001	9.84109256	9.57629541
1555267_at	79737	MULTMYEL	608.29999	9.24863917	8.88386512

Table 4.4: Example in DEAPP table of tranSMART

Key			Value
Row Key	Column Key		
	Family	Qualifier	
MULTMYEL + 79622	LOG	204454_at	6.16791991
MULTMYEL + 79622	LOG	216396_s_at	9.84109256
MULTMYEL + 79737	LOG	1555267_at	9.24863917

Table 4.5: Data sample from LOG Family HFile

Key			Value
Row Key	Column Key		
	Family	Qualifier	
MULTMYEL + 79622	RAW	204454_at	71.900002
MULTMYEL + 79622	RAW	216396_s_at	917.20001
MULTMYEL + 79737	RAW	1555267_at	608.29999

Table 4.6: Data sample from RAW Family HFile

Key			Value
Row Key	Column Key		
	Family	Qualifier	
MULTMYEL + 79622	ZSCORE	204454_at	8.306973
MULTMYEL + 79622	ZSCORE	216396_s_at	9.57629541
MULTMYEL + 79737	ZSCORE	1555267_at	8.88386512

Table 4.7: Data sample from RAW Family HFile

For example, if the raw intensity values of all patients in the trial MULTMYEL are retrieved, the Key Value system will locate the Key described in Table 4.8, and then load a HFile containing Key1, as shown in Table 4.9 into memory. This HFile probably also contains Key2, as shown in Table 4.10, because Key2 is typically stored adjacent to Key1 in the physical storage. This results in a reduction in query response time by increasing the cache hit rate.

Key		
Row Key	Column Key	
	Family	Qualifier
MULTMYEL + 00000	RAW	00000000

Table 4.8: Starting key for Key Value query

As in the relational model, Key Value data is organized as an LSM-tree. Each leaf in

Key		
Row Key	Column Key	
	Family	Qualifier
MULTMYEL + 79622	RAW	204454_at

Table 4.9: Query result Key1

Key		
Row Key	Column Key	
	Family	Qualifier
MULTMYEL + 79622	RAW	216396_s_at

Table 4.10: Query result Key2

the tree is a data block in a HFile (128 MB by default), each of which contains data from more than one patient. Considering the maximum probe set count in the database of 54,675 probe sets per sample using the Affymetrix HG-U133 chip, where the average probe set data size is 300 bytes,

$$300B * 54,675 \approx 16MB < 128MB \quad (4.2)$$

In this case, one type of probe set values for a patient can at most be stored in two HFiles. Therefore, each patient data in one type are stored within two HFiles. The time to fetch data of a patient  $t_p$  is:

$$t_p = 2 * p_s * \log_m(p_n) * t_{kv} \quad (4.3)$$

where the time to find a next node in the tree is one standard unit, the tree is of order  $m$ , the time to fetch a HFile in a Key Value system  $t_{kv}$ , and patient number is  $p_n$ . Thus, the total theoretical query time  $\sum t_{kv}$  is:

$$\sum t_{kv} = p_n * 2 * p_s * \log_m(p_n) * t_{kv} \quad (4.4)$$

In order to make a full comparison of time consumption between Key Value data model and relational data model,  $t_r$  is the time to load 32 KB (MySQL Clusters data page size) data from physical disk into memory and  $t_{kv}$  is the time to load 128 MB data, then in a common physical server with a SATA disk (X79 series chipset 6-Port SATA AHCI Controller),  $t_r$  is approximately 10 milliseconds and  $t_{kv}$  is approximately 9,000 milliseconds. In this case:

$$\sum t_r / \sum t_{kv} = (\log_m(p_n * p_s) * t_r) / (2 * \log_m(p_n) * t_{kv}) \quad (4.5)$$

Therefore, if there are 559 patients in a study and each patient has 54,675 probe sets, if cache and query optimizers are not taken into account, theoretically an ideal Key Value data model may be up to about 83 times faster than an ideal relational data model. The actual observed speedup is much less than the theoretical estimate since real-world performance will be different according to implementation. However, as the basis of the hypothesis, based on the theoretical calculations a Key Value will be significantly faster than a relational model when querying microarray data such as that stored in tranSMART.

### 4.2.2 Position Table

Position table provides high throughput queries over probe sets within a study. The STUDY\_NAME and PROBESET\_ID are used as the Row Key, as shown in Table 4.11. All records of a probe set in this study are saved together. If PROBESET\_IDs are required, these data will be loaded within very few disk reading commands. When multiple identical genetic attributes are retrieved, the composite key (STUDY\_NAME and PROBESET\_ID) can precisely locate the data by comparing this Row Key to the Start and End keys of a data block. Moreover, Family categorizes different types of data (RAW, LOG, and ZSCORE) into different data files to decrease the size of files to be searched.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
STUDY_NAME + PROBESET_ID	DATA_TYPE	SUBJECT_ID	EXPRESSION_VALUE
MULTMYEL + 204454 .at	RAW	79622	71.900002
MULTMYEL + 204454 .at	LOG	79622	6.16791991
MULTMYEL + 204454 .at	ZSCORE	79622	8.3069731

Table 4.11: Microarray position table

### 4.2.3 Cross-study Table

Cross-study table is utilized to search genetic attributes across studies. As shown in table 4.12, the STUDY\_NAME and PROBESET\_ID in the Row Key accelerate searches by gene id across studies. The composite Row Key (STUDY\_NAME and PROBESET\_ID) could improve the throughput of queries over several patients across many studies. All records of a probe set across studies are stored sequentially. When several PROBESET\_IDS in the whole database are retrieved, only a small number of the corresponding data files will be loaded.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
PROBESET_ID + STUDY_NAME	DATA_TYPE	SUBJECT_ID	EXPRESSION_VALUE
204454 .at + MULTMYEL	RAW	79622	71.900002
204454 .at + MULTMYEL	LOG	79622	6.16791991
204454 .at + MULTMYEL	ZSCORE	79622	8.3069731

Table 4.12: Microarray Cross-study table

## 4.3 Performance Evaluation

### 4.3.1 Experimental Environment

This dataset was transformed and loaded it into two different Key Value databases, HBase and MongoDB, and a relational database, MySQL Cluster, each of which was running on a virtual machine cluster. HBase was used to implement the relational model and the Key Value model, while MongoDB and MySQL Cluster both re-implement the relational model. Each database was configured as follows:

- HBase (Version 0.96.0 on Hadoop 1.0.3): One master server node and three slave nodes with HBase configured in fully distributed mode. The master server was configured as a virtual machine (VM) with 4 CPU cores and 8GB memory, while each slave node was configured as VMs with 2 CPU cores and 4GB memory. Each VM used a 100 GB disk. Three copies is the minimum number for the Hadoop Distributed File System to guarantee data consistency. Two types of queries in HBase are executed, Random Read and Scan, in order to select the faster method. In the Random Read method each patients data is retrieved discretely. In Scan, all patients from the first Row Key to the last Row Key expected are retrieved sequentially, including unexpected Row Keys between two expected ones. The Key Value model implementation follows the design in the previous section. The relational model implementation follows Figure 3.1, where Column position is Column PROBESET\_ID.
- MySQL Cluster (Version 5.6.11-ndb-7.3.2): A parallel database [DGG<sup>+</sup>86, FKT86] supports multi-dimensional queries and have good reliability. Four VMs are used, with one as a manager node and three data nodes. The manager node consists of a MGM, a MySQL and a NDB using a VM with 4 CPU cores and 8 GB memory. Each VM used a 100 GB disk. Each data node consisted of a NDB. Disk-based

InnoDB storage engine was used, where indexed columns and indices are stored in memory and non-indexed columns are stored in disks. The relational model implementation follows Figure 3.1, where Column position is Column PROBESET\_ID. The index uses BTREE [BM02, Com79] to speed up the query operations. Two data copies were used to guarantee its data consistency.

- MongoDB (Version 2.2.4): Four VMs were used in the MongoDB cluster (VM A configured with 4 CPU cores and 8GB memory, VMs B, C and D with 2 CPU cores and 4GB memory). Each VM used a 100 GB disk. VM A and B formed a replication set (rs0), with C and D forming another (rs1); rs0 and rs1 formed a two-shard sharding cluster. VMs B, C and D were also deployed as three configuration servers (mongod –configsvr). VM A hosted the load balancer (mongos), and handled all the data operation requests. MongoDB requires two copies to guarantee data consistency. The relational model implementation follows Figure 3.1, where Column position is Column PROBESET\_ID.
- Oracle (Version 11g): One VM was used with 16 CPU cores and 64 GB memory. The relational model implementation follows Figure 3.1, where Column position is Column PROBESET\_ID.

### 4.3.2 Query by subject

#### Benchmark generation

This experiment was performed using a dataset loaded in an instance of tranSMART running on a cloud computing test-bed, IC Cloud [GGT10], installed at Imperial College London. In the experiment a database, which tranSMART uses to store patient microarray data, was dumped and a public Multiple Myeloma (MULTMYEL) [RPB<sup>+</sup>09, HHZ<sup>+</sup>06] dataset [GEO:GSE24080] was extracted from the dump. The reason that MULTMYEL



was chosen as the test dataset was that it is one of the largest datasets loaded into the current tranSMART database instance, consisting of 559 samples and 54,675 probe sets for each sample, totaling approximately 30.5 million records.

In order to assess Key Value and Relational database model performances using a gradient of retrieval size requests, a series of typical marker selection queries have been devised based on relevant biological questions [PCG<sup>+</sup>10] and data download requests, as shown in Figure 4.6.

Case ID	Description	Subject number	Medical purpose
A1	Patients who underwent Therapy 2 and survived longer than 30 months	213	Discover gene expression patterns affecting the short term response to therapy 2
A2	Patients who underwent Therapy 2 and survived longer than 30 months	123	
B1	Patients who took Therapy 2 and were still alive at the end of the trial	459	Discover gene expression patterns affecting the long term response to therapy 2
B2	Patients who took Therapy 2 and did not survive the duration of the study	100	
C1	Patients who lived longer than 26 months	308	Discover gene expression patterns affecting patient survival
C2	Patients who survived less than 26 months	251	
D1	Patients who took Therapy 2	351	Discover the different effect of the two therapies
D2	Patients who took Therapy 3	208	
E1	Patients who survived less than 36 months	400	Discover gene expression patterns affecting patient survival
E2	Patients who survived more than 36 months	159	
F1	Patients who survived less than 52 months	496	Discover gene expression patterns affecting patient survival
F2	Patients who survived more than 52 months	63	
G	Whole dataset	559	Raw data download

Figure 4.6: Medical use cases

## Results

Each case was tested three times, where after every test the entire cluster was shut down to clean all caches to remove the influence of residual cached data.

The Figure 4.7 shows the comparison between the new Key Value design implemented in the HBase cluster and the original Oracle implementation used in the current transSMART. The Key Value implementation outperformed the Oracle one by average 18.17 times. Especially, when the query data set is small, Oracle performs more than 30 times slower than HBase.

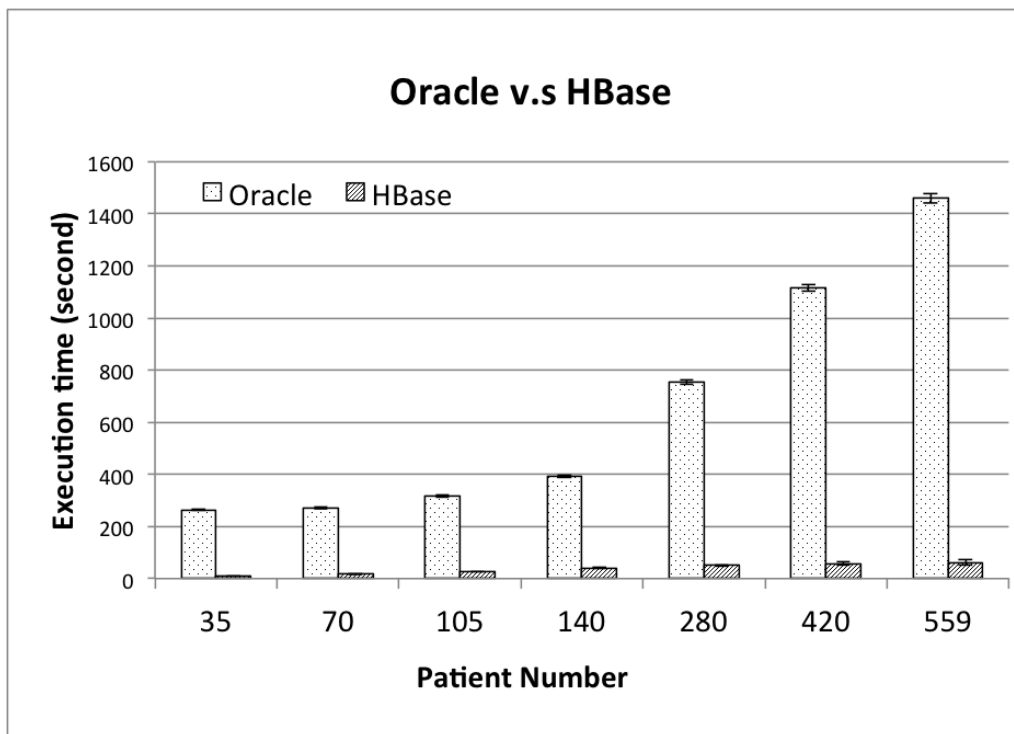


Figure 4.7: Performance of Key Value vs. original transSMART application. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries.

The record retrieval times for the HBase, MySQL Cluster and MongoDB configuration are shown in Figure 4.8. In this experiment only RAW data was targeted.

Compared to the relational model on HBase, in 5/11 of cases, the Key Value model per-

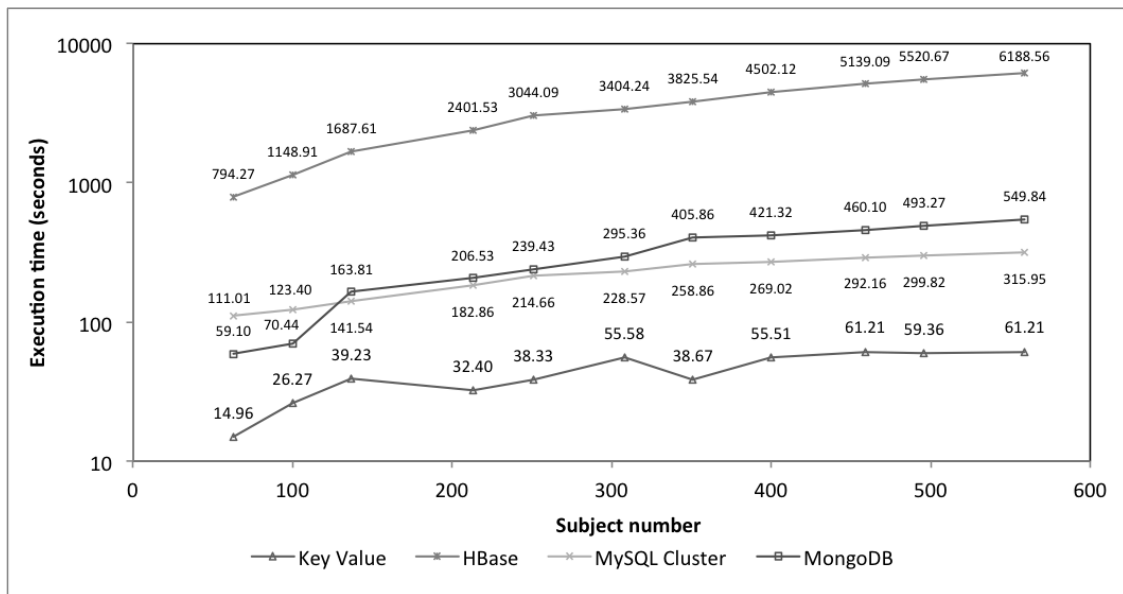


Figure 4.8: Performance of Key Value in HBase vs. relational data model in HBase, MySQL Cluster and MongoDB. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries.

forms more than 80 times faster. In 3/11 of cases, the Key Value model is more than 90 times faster. With the subject number increasing, the Key Value model advantage is more obvious. In the last and biggest case, the Key Value model outperforms the traditional one by more than 100 times.

With the number of records increasing, most test cases in relational model show retrieval times also increasing. MySQL Cluster retrievals are slow when querying fewer numbers of patient records with query times increasing slightly as the data queries scale up. the Key Value based data model demonstrates an average 5.24 times of increase compared to the relational model implemented on MySQL Cluster. In 6/11 of cases, the Key Value data model is more than 5 times faster than relational model on MySQL Cluster. In 10/11 of cases, the Key Value data model is more than 4 times faster than the relational model on MySQL Cluster. In the worst case, A2, the Key Value data model is more than 3.61 times faster than relational model on MySQL Cluster. As the size of the queried data scales upwards, MySQL Cluster performs less stable than other databases as show by the greater average result deviations in Figure 4.8.

The relational model on MongoDB performs well with smaller queries but slows significantly when the amount of data retrieved is scaled up. The Key Value based data model demonstrates an average 6.47 times increase in query performance compared to the relational model implemented on MongoDB. In 6/11 of cases, Key Value retrieval is more than 6 times faster than that on MongoDB, especially in cases with large retrieval data. In 10/11 of cases, the Key Value retrieval speed is more than 4 times faster than that on MongoDB. In the worst case, B2, the Key Value data model is more than 2.68 times faster than the relational model on MongoDB.

The data retrieval time in the Key Value model varies more widely due to the different read operations, Random Read and Scan, and also the Scan range in different cases, as shown in Figure 4.9. In most test cases in the experiment, Scans are quicker than Random Reads. For Scans, the worst case is when the expected patient count is small and the patient record distribution within the database is very sparse. In this situation, the Scan operation degrades to sequentially read all patient data. For example, in case B2, 63 patient records are expected and distributed over a range of 481 patients. However, a whole dataset Scan only needs about half of the time to perform a Random Read.

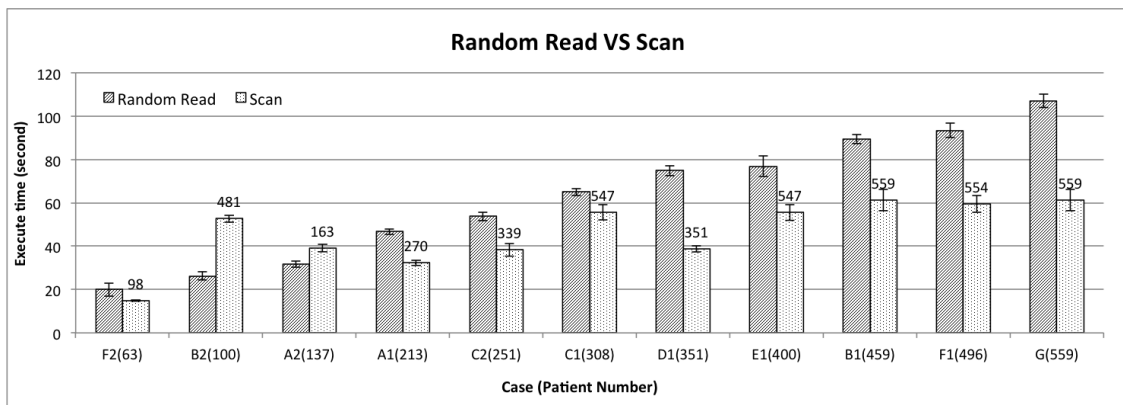


Figure 4.9: Performance of Random Read vs. Scan in Key Value data model. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries using both Random Read and Scan methods. The numbers above scan bar show the patient numbers the scan read in that case. The error bar shows the deviation of each test.

The situation is similar when retrieving other types of data, such as LOG or ZSCORE,

as shown in Figure 4.10 and 4.11. In these two figures, the performance trend observed for each Family is similar. Only the deviation values vary. Thus, compared to the other databases, the same conclusion will be generated for every Family.

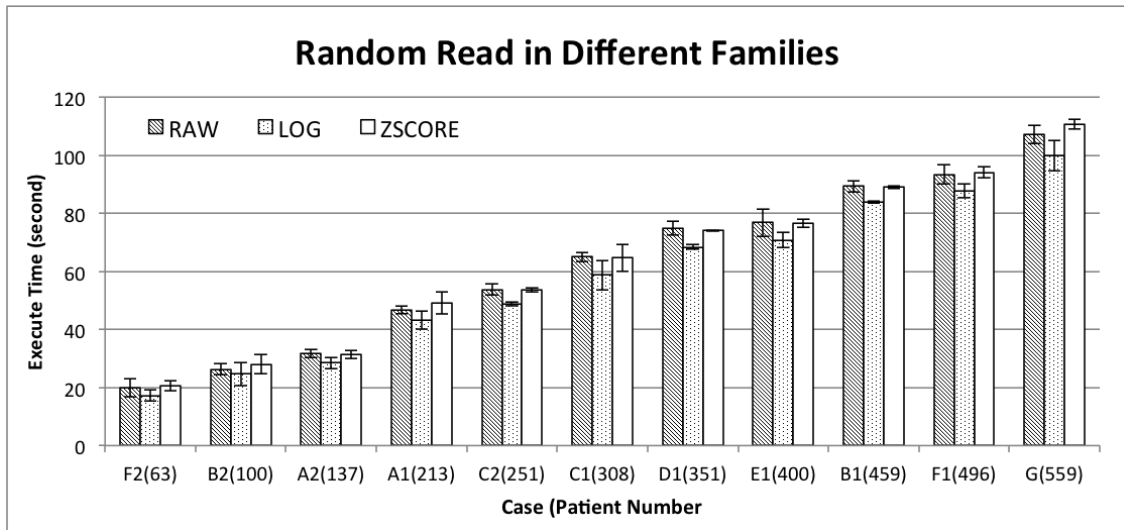


Figure 4.10: Performance of Random Read in different Families. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries of three Families. The error bar shows the deviation of each test.

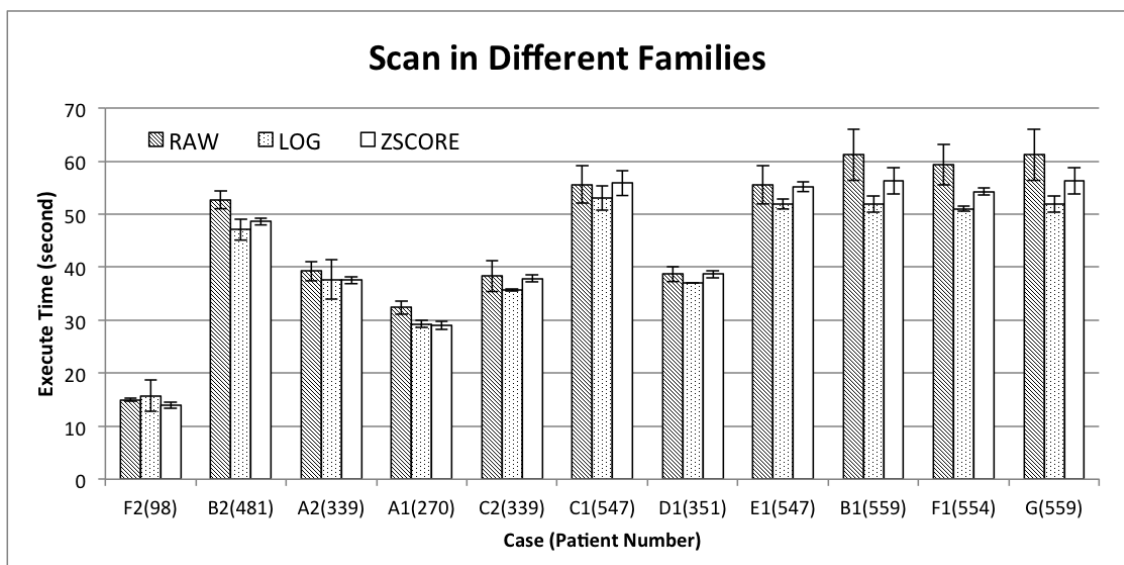


Figure 4.11: Performance of Scan in different Families. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries of three Families. The error bar shows the deviation of each test.

### 4.3.3 Query by probe within a study

#### Benchmark

A normal marker selection returns about 100 potential probe sets, which need to be verified by other datasets of similar studies. Based on the design microarray subject table will perform sub-standard for this purpose, but the microarray Position table can perform much faster. 100 probes are randomly generated to test the Key Value model against the relational one. The query of Key Value model is based on the Random Read operation.

#### Results

The Key Value data model demonstrates an average 174.52 times of increase compared to the relational model implemented on HBase. In 4/10 of cases, the Key Value model on HBase is more than 170 times faster than the relational one on HBase. In 9/10 of cases, the Key Value model on HBase performs 140 times faster than the relational model on HBase.

The Key Value data model demonstrates an average 10.19 times of increase compared to the relational model implemented on MongoDB. In 3/10 of cases, the Key Value model on HBase is more than 31 times faster than the relational one on MongoDB. In 6/10 of cases, the Key Value model on HBase performs 20 times faster than the relational model on MongoDB.

The Key Value based data model demonstrates an average 3.06 times of increase compared to the relational model implemented on MySQL Cluster. In 6/10 of cases, the Key Value data model is more than 3 times faster than relational model on MySQL Cluster. In 9/10 of cases, the Key Value data model is more than 2.5 times faster than the relational model on MySQL Cluster. In the worst case, 20-thread, the Key Value data model is more

than 2.47 times faster than relational model on MySQL Cluster.

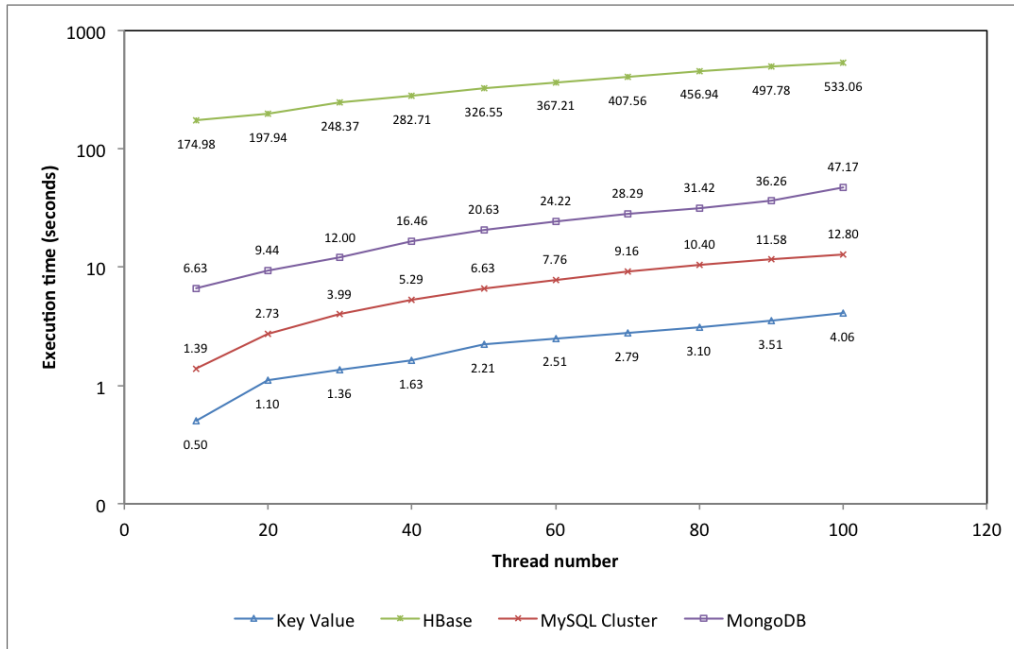


Figure 4.12: Query by position within one dataset. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries. The error bar shows the deviation of each test.

#### 4.3.4 Query by probe across studies

##### Benchmark

A further validation was performed using three GSE24080 studies. The GSE24080 dataset is copied twice and renamed GSE24081 and GSE24082. The 100 probe sets in the query by position experiment are reused. MySQL Cluster failed to load these three datasets due to a memory limitation. Thus, the Key Value model on HBase is tested against the relational model on HBase.

## Results

Before this experiment, the heaviest 100-thread concurrent query has been used to test which is the better operation for Key Value data model, Range Scan or Random Read. The Range Scan (8.480 seconds) outperformed the Random Read (13.320 seconds) by 1.57 times. With dataset number increasing, Range Scan should perform even better than Random Read due to the Bigtable design. Thus, all the queries in these three datasets are based on the Range Scan operation.

The Figure 4.13 shows the Key Value data model on HBase demonstrates an average 246.48 times of increase compared to the relational model on HBase, as shown in Figure 7. In 6/10 of cases, The Key Value model on HBase is more than 220 times faster than the relational model on HBase. In 3/10 of cases, the Key Value data model is more than 250 times faster than the relational model on HBase. In largest case, the Key Value data model is more than 210 times faster than the relational model on HBase.

The Key Value data model on HBase demonstrates an average 27.67 times of increase compared to the relational model on MongoDB, as shown in Figure 7. In all cases, The Key Value model on HBase is more than 20 times faster than the relational model on MongoDB. In the last two largest cases, the Key Value data model is more than 30 times faster than the relational model on MongoDB.

## 4.4 Implementation on tranSMART

### 4.4.1 Implementation

This implementation is based on HBase 0.98.0. As shown in the Figure 2.11, trial, patient and annotation information are retrieved from a SQL database before a high dimensional



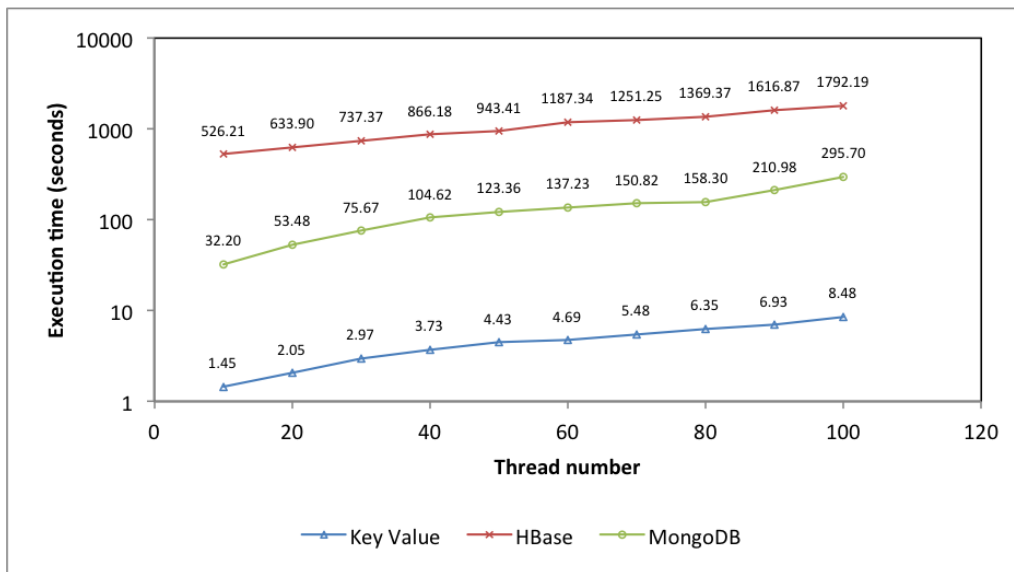


Figure 4.13: Query by position across three datasets. The curves show the query retrieval times for each of the test cases over varying numbers of patient record queries. The error bar shows the deviation of each test.

data query. The high dimensional data query function first connects to HBase microarray table and then search the expected Key Value records according to the query conditions, as shown in Figure 4.14. The first step is establishing a connection to the HBase microarray table and setting the data type. Then overridden `getRecord` functions are used to send data query requests. There are two kinds of requests in the code, as shown in Figure 4.15. For one thing, if no genetic positions are specified, all available genes will be returned. For another, if the gene information is specified, these genes will be used as filters.

```

/* connect to HBase microarray table,
 * dataType is used to specify the data types,
 * such as RAW, LOG, ZSCORE. */
KVModule kvaModule = connectTo("microarray-table", dataType)
if (geneList == null) {
    /* if no genetic positions are specified,
     * all available genes are included. */
    kvResults = kvMrnaModule.getRecord(trialName, patientList)
} else {
    /* if genetic positions are specified,
     * a gene to probe map is included as a filter. */
    kvResults = kvMrnaModule.getRecord(trialName, patientList,
                                       gene2probeMap)
}

```

Figure 4.14: Pseudocode of calling microarray data query functions.

```

/**
 * retrieve data from HBase microarray table
 * @parameter trialName: the name of the trial(study)
 * @parameter patientList: the list of patients
 * @parameter dataType: the required data type, such as RAW, LOG, ZSCORE
 */
getRecord(trialName, patientList) throws IOException {
    return getRecord(trialName, patientList, null, dataType);
}

/**
 * retrieve data from HBase microarray table
 * @parameter filterList: the list of specified genes of probes
 */
getRecord(trialName, patientList, filterList) throws IOException {
    List results;
    for (patientID : patientList) {
        results.addAll(getKV(trialName, patientID));
    }
    return results;
}

/**
 * retrieve Key Value pairs from HBase microarray table
 */
getKV (trialName, patientID, filterList) {
    List result;
    Get g = new Get(trialName + ":" + patientID);
    g.addFamily(dataType);
    if (filterList != null) {
        g.setFilter(filterList);
    }
    MicroarrayTable.setScannerCaching(cacheSize);
    Result r = MicroarrayTable.get(g);
    for (Cell kvpair : r.rawCells()) {
        result.add(kvpair);
    }
    return result;
}

```

Figure 4.15: Pseudocode of retrieving microarray data from HBase.

## 4.4.2 Performance evaluation

The data exportation, as shown in Figure 4.16 and 4.17, is one of the most time consuming services in tranSMART. This service exports all the data related to a patient, such as clinical and high dimensional data. This service in tranSMART v1.2 research branch was used to export microarray based data and test the performance against an actual use case U-BIOPRED. All the data in the U-BIOPRED data were loaded into a tranSMART PostgreSQL database. HBase data model was implemented based on this tranSMART application and the same microarray data were loaded into the HBase database.

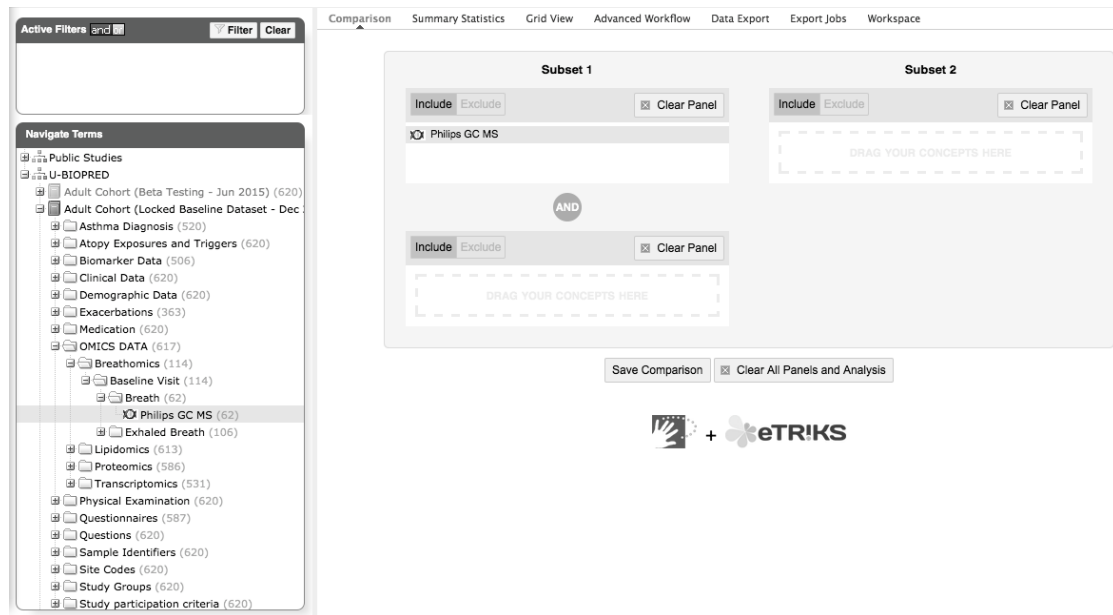


Figure 4.16: Data export service in tranSMART (select patient set).

## Experimental Environment

This dataset was transformed and loaded it into HBase, which was running on OpenStack. HBase was used to implement the Key Value model, while PostgreSQL implemented the relational model. Each database was configured as follows:

- HBase (Version 0.98.0 on Hadoop 2.5.2): One master server node and three slave nodes with HBase configured in fully distributed mode. The master server was configured as a VM with 4 CPU cores and 4GB memory, while each slave node was configured as VMs with 4 CPU cores and 4GB memory. Each VM used a 100 GB disk. Three copies is the minimum number for the Hadoop Distributed File System to guarantee data consistency.
- PostgreSQL (Version 9.4.1): One VM was used with 16 CPU cores and 16 GB memory, on which a standard tranSMART configuration was applied.

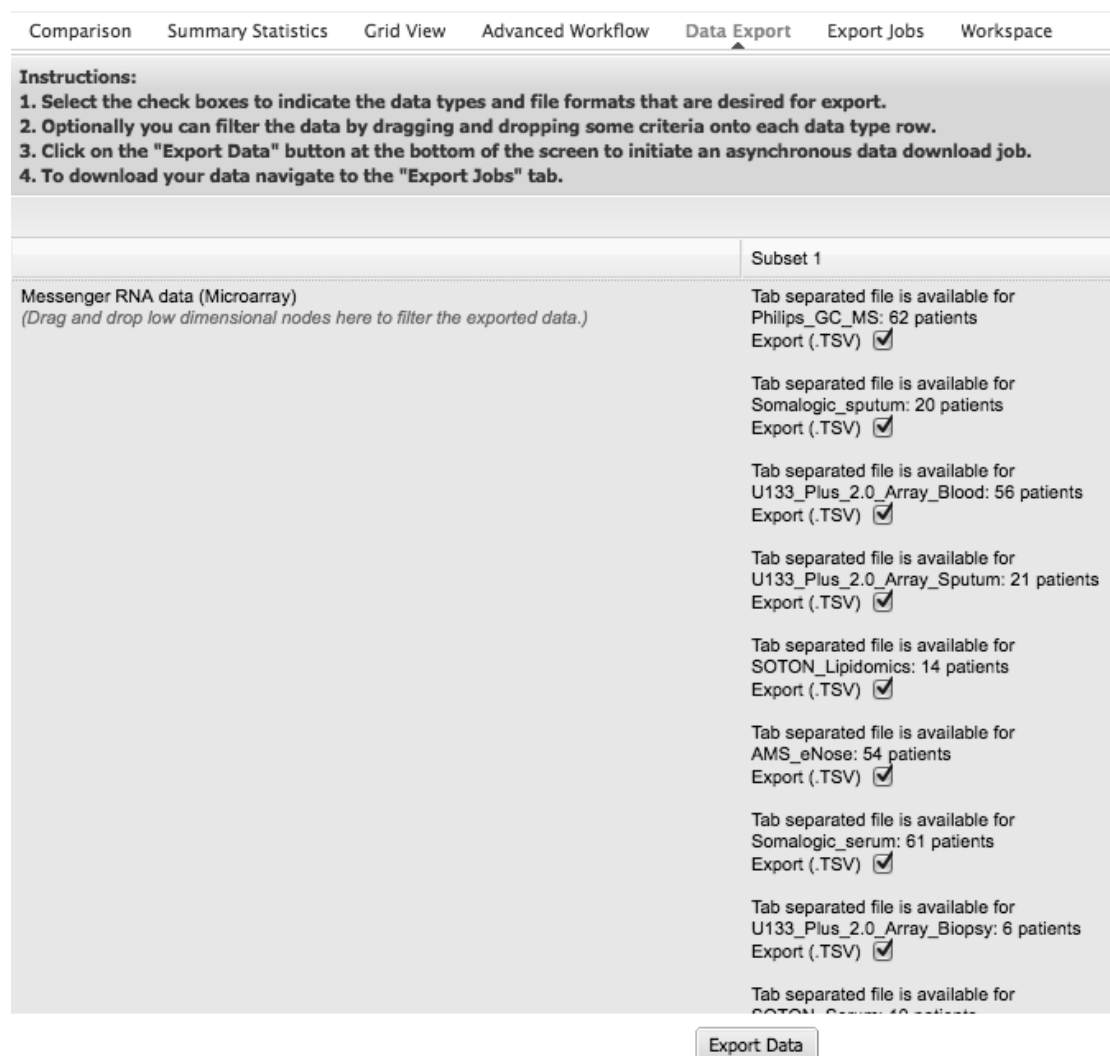


Figure 4.17: Data export service in tranSMART. (select high dimensional data set).

## Benchmark generation

A dataset from U-BIOPRED project, which contains more than 55 million genomic expression records, is one of the biggest use cases that have been stored in tranSMART. The U-BIOPRED project aims to speed up the development of better treatments for patients with severe asthma. Several knowledge gaps today make it hard to predict in the early stages of drug development how well a new experimental medicine will work in patients. One of the major difficulties is the finding that there are many different forms of severe asthma, caused by different mechanisms of disease. Patients with different types of

asthma may react differently to new or existing treatments. Clustering patients' genomic expression data is a widely used method to identify different types of asthma.

In the dataset of June 2014 there are 620 patients participated and 12 types of microarray data collected, including breathomics, lipidomics, proteomics and transcriptomics, totally more than 55 million expression values. A patient may took different microarray tests. For example, the breathomics data contains 2 datasets, Philips\_GC\_MS and Amsterdam\_eNose. 62 samples are tested in Philips\_GC\_MS and each patient has 7,039 probe tests. 106 patients are tested in Amsterdam\_eNose, each with 190 probes. One dataset in lipidomics contains 128 samples, each with 15,622 probe sets. Proteomics data are the largest, which contains 4 datasets, Somalogic\_serum(579 samples, 1129 probe sets), Soton\_serum(131 samples, 138 probe sets), Somalogic\_sputum(121 samples, 1129 probe sets) and Sotom\_sputum(100 samples, 3551 probe sets). Table 4.18 lists all the details about all these datasets.

category	dataset	sample	probe set	record
breathomics	philips	62	7039	436418
	amsterdam enose	106	190	20140
lipidomics	lipid	128	15,622	1999616
proteomics	somalogic serum	579	1129	653691
	soton serum	131	138	18078
	somalogic sputum	121	1129	136609
	soton sputum	100	3551	355100
transcriptomics	blood	485	54715	26536775
	nasal brushings	89	54715	4869635
	sputum	120	54715	6565800
	bronchial biopsy	108	54715	5909220
	bronchial Brushings	149	54715	8152535

Figure 4.18: U-BIOPRED dataset structure.

Three sets of samples (Philips\_GC\_MS, Somalogic\_serum and GPL570\_Blood) in a tranSMART instance, which is held in OpenStack, are chosen to test the performance of different sizes of sample set, as shown in Figure 4.19. In each test all probe set data of each sample will be fetched from a database into memory. For example, if sample\_A has probe set data in Philips\_GC\_MS, Amsterdam\_eNose and Soton\_serum, all these data will be

retrieved in this query.

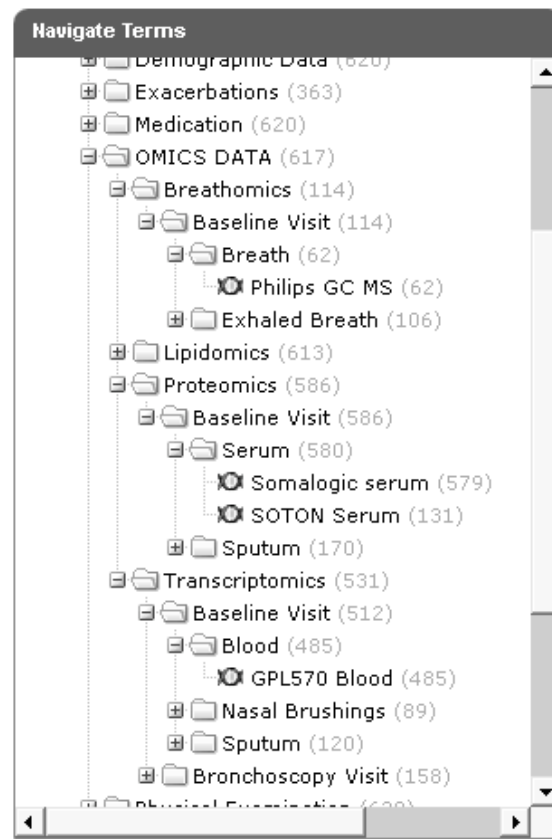


Figure 4.19: tranSMART instance with U-BIOPRED data.

## Results

In Figure 4.20 the HBase implementation greatly outperformed the PostgreSQL by average 6.36-fold speedup. In the Philips\_GC\_MS case, HBase performed 7.64 times faster than PostgreSQL, whilst in Somalogic\_serum and GPL570\_Blood cases HBase showed 5.90 and 5.54-fold speedup respectively.

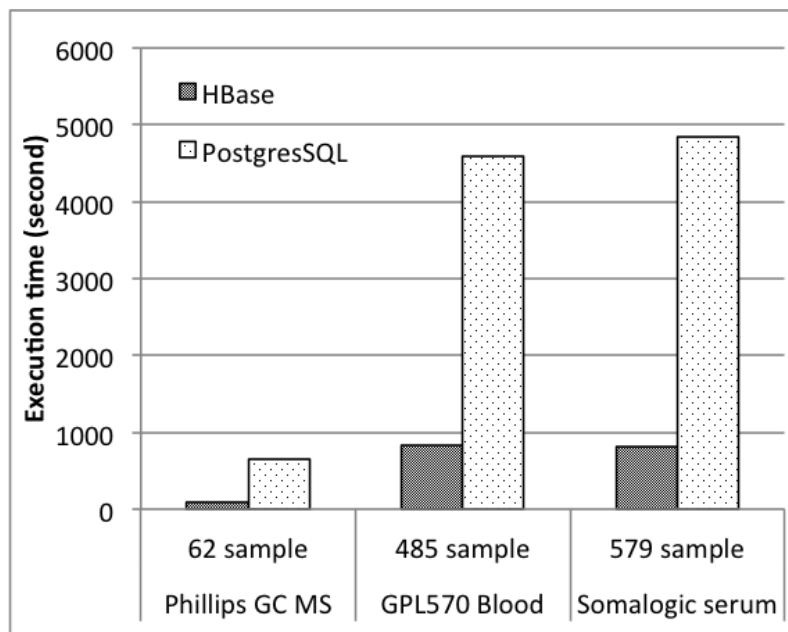


Figure 4.20: Performance evaluation on U-BIOPRED data. The bar chart shows the query retrieval times for each of the test cases over varying numbers of patient record queries. The bottom line shows the name of the item in the clinical tree in Figure 4.19

## 4.5 Discussion

The results show that in general the microarray Key Value implementation of the transMART DEAPP schema using HBase outperforms the relational model on HBase, MySQL Cluster and MongoDB. HBases increased performance in comparison to MySQL Cluster comes from the new Key Value data model.

Queries in HBase are optimized manually by choosing to perform either a pure Scan or pure Random Read operation. The current implementation is based on the pure Random Read operation. Ideally there should be a query optimizer built into the HBase system, like in MySQL Cluster or MongoDB, to automatically generate query plans for high-dimensional data retrieval. Pure Scans or Random Reads may not be the best choice for certain cases, where a mixed, dynamically selected query method may perform better than statically choosing one or the other.

The results also show that performance in different Families differ slightly. Family LOG

performs best in almost all cases. This phenomenon may result from the LOG HFile location in HBase system. For example, the LOG data are evenly distributed in all the VMs and all these VMs can serve for LOG data query concurrently.

The relational model implementation consumes a lot of memory. Although MySQL Cluster supports disk based storage, all indexed columns and indices are still stored in memory. This feature may influence its scalability. Other SQL-based database cluster, such as PostgreSQL XE and Oracle Clusterware [GCK<sup>+</sup>10], may be better for further evaluation.

The relational model in MySQL Cluster performs more stable than the Key Value model in HBase. All the deviations of the relational model in MySQL Cluster are less than 11% of the execution time, but the Key Value model in HBase has deviations larger than 20%. For example, the 20 thread Key Value deviation in Figure 4.12 is 23% of the execution time. The figure shows the average execution time (1.10 seconds) and the three raw execution times are 0.92, 1.02, 1.37 seconds. This large deviation mainly results from the last longest query time (1.37 seconds). Internal and external factors may affect the query performance. Internal communication, such as heartbeats and schedules, synchronizes workers and distributes tasks frequently. Also, loggers record database behaviour in memory before any action actually happens and flush the information into permanent storage periodically. If any worker in a cluster is much busier than others, the logger in this worker may flush data more often than others. Loggers in different workers record the corresponding worker activities asynchronously. Some Logger may suffer from timeout during I/O busy time. The server will throw an exception for the timeout and restart the Logger. During this period, the server cannot response any requests. In addition, HBase data locality may be changed from time to time. For example, each time a region may be assigned to a different region server. This is due to the periodical HBase maintenance operation, such as region splitting and region merging. The external influence mainly comes from the VM hypervisor I/O scheduler. The VM hypervisor controlled all the guest VMs in a physical server. Any VM requests are handled by the hypervisor and transformed to



the physical system. If other VMs share the physical machine at the same time, requests from all VMs will be sent to the same queue. The same request in a VM may be executed for a longer time if other VMs send many requests at that moment. The data block in HBase (128 MB) is much larger than that in MySQL Cluster (16 kb), so HBase requires more network resources, especially for random read tests. If the network becomes busy, HBase query response time is easier to be affected. Thus, if suffering from any of such interruptions, the HBase standard deviation will be quite large.

## 4.6 Conclusion

This chapter introduces a Key Value data model to improve the performance when querying microarray data, compared to the relational model currently implemented in transSMART. In the performance evaluation experiments, the Key Value data model implemented in HBase outperformed the relational model implemented in HBase, MySQL Cluster and MongoDB by up to 246, 7 and 20 times. When integrated into transSMART this Key Value implementation performs up to 7 times faster than the current transSMART.

# Chapter 5

## SNP Key Value Model

### 5.1 Introduction

A Single Nucleotide Polymorphism, also known as SNP, is a DNA sequence variation occurring commonly within a population in which a single nucleotide, A, T, C or G, in the genome (or other shared sequence) differs between members of a biological species or paired chromosomes. For example, two sequenced DNA fragments from different individuals, TTCGGAT v.s TTCGAAT, contain a difference in a single nucleotide, -G- v.s -A-. In this case there are two alleles. Almost all common SNPs have only two alleles. The genomic distribution of SNPs is not homogenous. SNPs occur in non-coding regions more frequently than in coding regions. In general, natural selection is eliminating other variants of the SNP that constitutes the most favorable genetic adaptation. Other factors, such as genetic recombination and mutation, can also affect SNP density.

SNP density can be predicted by the occurrence of microsatellites [TPA<sup>+</sup>00]. For example, AT microsatellites in particular are crucial predictors of SNP density, with a long repeated AT sequence tending to be found in regions, where SNP density are significantly reduced.

Within a human population, SNPs can be assigned the lowest allele frequency at a locus observed in this population. There are variations between populations, so a SNP allele, which is general in one geographical or ethnic group, may be rare in another.

These genetic variations between individuals, especially in non-coding parts of the genome, are often used in DNA fingerprinting [KS<sup>+</sup>98]. This feature has been exploited in forensic science. Also, these genetic variations underlie differences in the susceptibility to disease. The severity of illness and how the body responds to treatments are also manifestations of genetic variations.

Single-nucleotide polymorphisms may fall within coding sequences of genes, non-coding regions of genes, or in the intergenic regions (regions between genes). SNPs within a coding sequence do not necessarily change the amino acid sequence of the protein that is produced, due to degeneracy of the genetic code.

SNPs in the coding region are of two types, synonymous and nonsynonymous SNPs [RBS02]. Synonymous SNPs do not affect the protein sequence while nonsynonymous SNPs change the amino acid sequence of protein. The nonsynonymous SNPs have two types: missense and nonsense.

Variations in the DNA sequences of humans can affect how humans develop diseases and respond to pathogens, chemicals, drugs, vaccines, and other agents. SNPs are also critical for personalized medicine [HC10]. However, their greatest importance in biomedical research is for comparing regions of the genome between cohorts in genome-wide association studies.

The study of SNPs is also important in crop and livestock breeding programs. SNP genotyping shows details on the different approaches used to identify SNPs, where the genotype greatly affects a specific characteristic (phenotype) of the cell, organism or individual. DNA mutations that are acquired rather than inherited, such as cancer mutations, are not part of the individual's genotype.

SNPs have been used in genome-wide association studies GWAS [SSR11]. The knowledge of SNPs helps in understanding pharmacokinetics or pharmacodynamics, i.e. how drugs act in individuals with different genetic variants. Diseases with different SNPs may become relevant pharmacogenomic targets for drug therapy. Some SNPs are associated with the metabolism of different drugs.

A wide range of human diseases, e.g. Sicklecell anemia [SS92], Thalassemia and Cystic fibrosis [KRB<sup>+</sup>89] result from SNPs. SNPs without an observable impact on the phenotype, which is usually named silent mutations, are still useful as genetic markers in genome-wide association studies, because of their quantity and the stable inheritance over generations.

For complex diseases, rather than functioning individually, SNPs usually work in coordination with other SNPs to manifest a disease condition.

Thousands of GWAS of significant human diseases have been conducted recently. Many of such studies use SNP arrays, for example in the International HapMap project [Con07, The10, The05] that aims to develop a map describing the common patterns of human genetic variation. The SNP data has already benefited identification of disease subgroups and the prediction of responses of individual subjects, named molecular profiling based patient stratification. Biomedical research is moving towards using high-throughput variant data to improve clinical decision making. One approach for building classifiers is to stratify subjects based on their variants. Unsupervised clustering algorithms can be used for stratification purposes.

The motivation for optimization of SNP data model is based on the experiences in using tranSMART. The aim is to optimize tranSMART so that clinicians can make use of it for faster and more confident clinical decision making. However, the current tranSMARTs I/O performance of the SNP data is sub-standard.

In response to a requirement for a general genome variation map required by association

studies, gene mapping and evolutionary biology, NCBI has created the dbSNP database [SWK<sup>+</sup>01] and VCF file format [DAA<sup>+</sup>11]. HapMap project uses another genotype data format similar to VCF. Though VCF and HAPMAP files are easy-to-use, they have crucial issues with data scalability, concurrent data access speed and fault tolerance. Relational database implementations, such as Oracle, PostgreSQL and MySQL Cluster, to store VCF data overcome most of the disadvantages of flat files. TranSMART has developed a relational model similar to dbSNP. Furthermore, SeqWare introduces a Key Value based data model that performs much faster than the relational ones. Although SeqWare greatly outperformed relational models implemented in BerkeleyDB on queries by genetic positions (Row Key), CCIndex indicates the low-efficient Qualifier query due to the inefficient HBase secondary index [LY01], which SeqWare utilizes to accelerate queries by variant (Qualifier). Index scan over the original table using random reads is inefficient. The following of this section introduces each model mentioned above.

### 5.1.1 dbSNP

Though SNP is the abbreviation for single nucleotide polymorphism, dbSNP database schema is a public data model for all kinds of short sequence variation. dbSNP contains a broad collection of simple genetic variations and classifies nucleotide sequence variations with the following types, such as single base nucleotide substitutions [NTT97], small-scale multi-base deletions or insertions polymorphisms [RHAG<sup>+</sup>90, RHCS92, WDH<sup>+</sup>02], microsatellite repeats [TPA<sup>+</sup>00, TS93, CLH<sup>+</sup>99], named variants [Car58], and uncharacterized heterozygous assays [IFY<sup>+</sup>93].

The dbSNP is designed for research into a wide range of translational medical problems that include the identification of genotype-phenotype relationships [MC00], genetic and physical mapping [PSH<sup>+</sup>98, CBW<sup>+</sup>97], functional analysis [ZC96], pharmacogenomics, and association studies.

Data in dbSNP can be from any organism, from any part of a genome, and can include genotype and allele frequency data. dbSNP accepts data for all classes of simple sequence variation, and provides access to clinically significant variations of germline or somatic origin. Individual genotypes may be loaded for any variation. dbSNP accepts individual genotypes from samples provided by public databases, such as HapMap or the 1000 Genomes project. Genotypes in dbSNP include links to method descriptions and population. General genotype data offer the foundation for individual haplotype definitions. General genotype data are also widely used for selecting positive and negative control reagents in new experiments, further analysis in genome association studies [LRPR98], and testing and refining haplotype reconstruction algorithms [PGL99].

There is no requirement on minimum allele frequencies or functional neutrality for the polymorphisms in the database. Thus, the scope of dbSNP includes disease-causing clinical mutations as well as neutral polymorphisms.

The dbSNP individual genotype data schema is a widely used relational data model for SNP data, which is originally designed to store data from different types of short sequence variation. Given the complexity of the full dbSNP database schema, the main data tables are extracted to explain the genotype model, as shown in Figure 5.1. The dbSNP stores this data mainly in four tables, the SNP, individual, genotype and observation tables. In addition to these four main tables, there are many tables used to accurately describe their details. The four data tables, along with the detailed tables make up the star schema of the warehouse, with the central observation table surrounded radially by SNP, individual and genotype tables. The most important concept regarding the construction of a star schema is identifying what constitutes a fact.

The observation table (SubInd) contains the links (identifiers) to SNP, individual and genotype tables. The SubInd table is the central table of the dbSNP star schema and represents the intersection of the other tables. Each row describes one observation about

a genotype of a SNP in an individual. Most queries in the dbSNP database require joining the SubInd table with one or more other tables together, such as SNP individual or genotype.

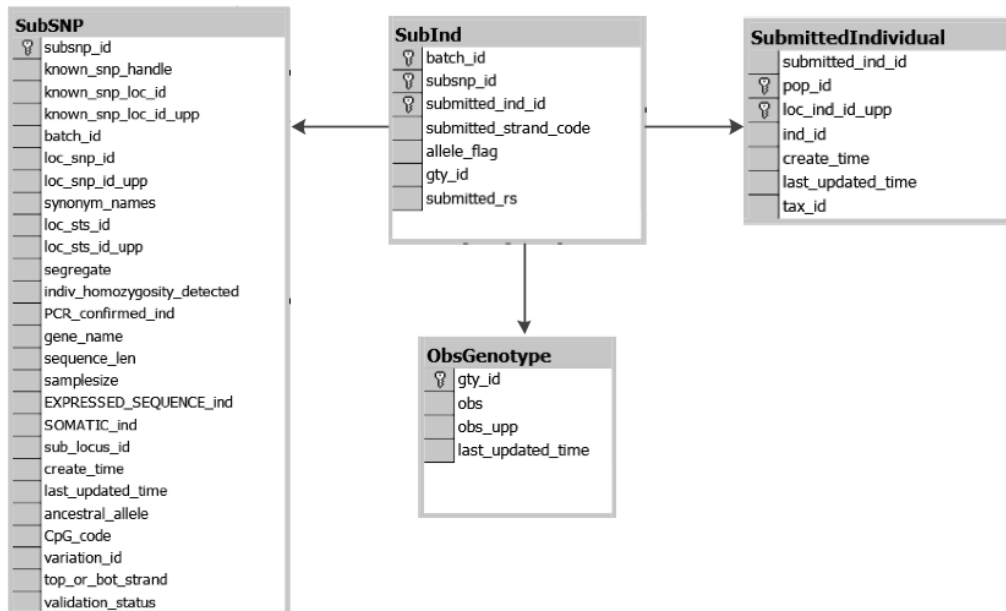


Figure 5.1: dbSNP data model

### 5.1.2 VCF format

The VCF is a widely used open source data model for gene sequence variation data introduced by the VCF tools project with significant contributions from the 1000 Genomes project [ALA10]. VCF accepts different sorts of variation data such as single nucleotide variants, insertions/deletions, copy number variants and structural variants.

As shown in Figure 5.2, file meta-information is represented by comma-delimited key=value pairs after the ## string. These meta-information lines usually describe the INFO, FILTER and FORMAT entries used in the body of the VCF file. The header line names the 8 fixed and mandatory columns, including #CHROM, POS, ID, REF, ALT, QUAL, FILTER and INFO. All data lines are tab-delimited. The missing values are represented by a dot.

```

##fileformat=VCFv4.1
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x">
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA00001 NA00002 NA00003
20 14370 rs6054257 G A 29 PASS NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1|1:43:5:..
20 17330 . T A 3 q10 NS=3;DP=11;AF=0.017 GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3 0|0:41:3
20 1110696 rs6040355 A G,T 67 PASS NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2 2|2:35:4
20 1230237 . T . 47 PASS NS=3;DP=13;AA=T GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:48:4:51,51 0|0:61:2
20 1234567 microsat1 GTC G,GTCT 50 PASS NS=3;DP=9;AA=G GT:GQ:DP 0|1:35:4 0|2:17:2 1|1:40:3

```

Figure 5.2: VCF data format

- CHROM, short for chromosome, is an identifier from the reference genome or an angle-bracketed ID String pointing to a contig in the assembly file.
- POS, short for position, is the reference position, with the 1st base having position 1.
- ID, short for identifier, is a semi-colon separated list of unique identifiers where available.
- REF, short for reference base.
- ALT, short for alternate base, is a comma-delimited list of alternate non-reference alleles called on at least one of the samples.
- QUAL, short for quality, is phred-scaled quality score for the assertion made in ALT.
- FILTER stands for filter status. If this position has passed all filters, FILTER value is PASS; otherwise, a semicolon-delimited list of codes for filters that fail is present.
- INFO is short for additional information.
- FORMAT: If genotype information is included, the FORMAT field specifies the data types and order (colon-separated alphanumeric String). The FORMAT is fol-



lowed by one field per sample, with the colon-separated data in this field corresponding to the types specified in the FORMAT. The first sub-field GT (genotype) is present firstly, if it exists. No sub-field is mandatory.

### 5.1.3 HapMap

The HapMap is a data warehouse for genotype data produced by International Haplotype Map Project [Ade07]. HapMap genotype uses another genotype data format very similar to VCF. The data files contain genotypes on all SNPs produced by HapMap genotyping centers and other independent sources, such as Affymetrix and Illumina, and submitted to the HapMap Data Coordination Center. As shown in Figure 5.3, genotype files includes tab-delimited flat files with at least the following fields:

- rs#: refSNP identifier at the time of release
- SNPalleles: SNP alleles according to dbSNP
- chrom: chromosome that SNP maps to
- pos: chromosome position of SNP, in base pairs on reference sequence
- strand: strand of reference sequence that SNP maps to
- genome\_build: version of reference sequence assembly
- center: HapMap genotype center that produced the genotypes
- protLSID: LSID for HapMap protocol used for genotyping
- assayLSID: LSID for HapMap assay used for genotyping
- panelLSID: LSID for panel of individuals genotyped
- QC\_code: QC-code, currently 'QC+' for all entries

- all following fields: observed genotypes of samples, one per column, sample identifiers in column headers (Coriell catalog numbers, example: NA10847). Duplicate samples have .dup suffix.

```

rs# SNPalleles chrom pos strand genome_build center protLSID assayLSID panelLSID QC_code NA06985 NA06991 NA06993
NA06994 NA07000 NA07019 NA07022 NA07029 NA07034 NA07048 NA07055 NA07056 NA07345 NA07348 NA07357 NA10830 NA10831
NA10835 NA10838 NA10839 NA10846 NA10847 NA10851 NA10854 NA10855 NA10856 NA10857 NA10859 NA10860 NA10861 NA10863
NA11829 NA11830 NA11831 NA11832 NA11839 NA11840 NA11881 NA11882 NA11992 NA11993 NA11994 NA11995 NA12003 NA12004
NA12005 NA12006 NA12043 NA12044 NA12056 NA12057 NA12144 NA12145 NA12146 NA12154 NA12155 NA12156 NA12234 NA12236
NA12239 NA12248 NA12249 NA12264 NA12707 NA12716 NA12717 NA12740 NA12750 NA12751 NA12752 NA12753 NA12760 NA12761
NA12762 NA12763 NA12801 NA12802 NA12812 NA12813 NA12814 NA12815 NA12864 NA12865 NA12872 NA12873 NA12874 NA12875
NA12878 NA12891 NA12892
rs3016036 A/T Chr22 14430252 + ncbi_b34 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
urn:lsid:perlegen.hapmap.org:Assay:25760.102001:1 urn:lsid:ccc.hapmap.org:Panel:CEPH-30-trios:1 QC+ AA AA AA AA AA
AT AT AA AT AT AA AT AT AT AT AA AA AA AA AT AT AA AA AA NN AT AA AA NN AA AT AT AA AA AT AA AT AA AA NN AT AA AA
AA AT AA AT AA AA AA AT NN AT AT AA AT NN AT AA AA AA AA AA AT AA AA AA AA AA AA AT AA AA AT AA AT AA AA AA AA NN
AA AA AA NN AT AT AA
rs2334386 G/T Chr22 14430353 + ncbi_b34 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
urn:lsid:perlegen.hapmap.org:Assay:25760.6364532:1 urn:lsid:ccc.hapmap.org:Panel:CEPH-30-trios:1 QC+ GG GG GG GT GG
GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG
GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG
rs2844882 A/G Chr22 14430375 + ncbi_b34 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
urn:lsid:perlegen.hapmap.org:Assay:25760.7708542:1 urn:lsid:ccc.hapmap.org:Panel:CEPH-30-trios:1 QC+ AA AA AA AA
AA AA AA AA AA AA AA AA AA AG AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
AA AA AA AA AA AA AA AA AG AA AA AA AG AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
AA AA AA AA AA AA AA

```

Figure 5.3: HapMap data format

### 5.1.4 TranSMART SNP model

TranSMART currently adopts the relational implementation. Two main tables in the SNP database schema, `SNP_INFO` and `VARIANT_SUBJECT_DETAIL`, as shown in Table 5.1 and 5.2, store SNP description data and subject variant details respectively. The speed of SNP data queries has been observed as a bottleneck due to the increasing amount of SNP data stored and processed in tranSMART.

However, with data accumulated from high-throughput NGS, relational databases are unlikely to meet high-performance requirements as the magnitude of datasets scale upwards. Key Value databases, such as HBase, MongoDB and Hypertable, have become commonplace in recent years providing better scalability and faster data access.

Column	Type
VARIANT_SUBJECT_DETAIL_ID	NUMBER (9,0)
SNP_INFO_ID	NUMBER (22,0)
QUAL	VARCHAR2 (100 BYTE)
ENTREZ_ID	VARCHAR2 (50 BYTE)
FILTER	VARCHAR2 (50 BYTE)
INFO	VARCHAR2 (2000 BYTE)
FORMAT	VARCHAR2 (50 BYTE)
VARIANT_VALUE	CLOB

Table 5.1: TranSMART SNP\_INFO table

Column	Type
DE_RC_SNP_INFO	VARCHAR2 (100 BYTE)
SNP_INFO_ID	NUMBER (22,0)
RS_ID	NVARCHAR2 (50)
CHROM	VARCHAR2 (4 BYTE)
POS	NUMBER (10,0)
REF	VARCHAR2 (1000 BYTE)
ALT	VARCHAR2 (1000 BYTE)
GENE_INFO	VARCHAR2 (1000 BYTE)
VARIATION_CLASS	VARCHAR2 (10 BYTE)
STRAND	VARCHAR2 (1 BYTE)
CLINSIG	VARCHAR2 (100 BYTE)
DISEASE	VARCHAR2 (500 BYTE)
GMAF	NUMBER (10,0)
GENE_BIOTYPE	VARCHAR2 (100 BYTE)
IMPACT	VARCHAR2 (50 BYTE)
TRANSCRIPT_ID	VARCHAR2 (100 BYTE)
FUNCTIONAL_CLASS	VARCHAR2 (100 BYTE)
EFFECT	VARCHAR2 (100 BYTE)
EXON_ID	VARCHAR2 (100 BYTE)
AMINO_ACID_CHANGE	VARCHAR2 (50 BYTE)
CODON_CHANGE	VARCHAR2 (100 BYTE)
HG_VERSION	VARCHAR2 (10 BYTE)
GENE_NAME	VARCHAR2 (50 BYTE)
ENTREZ_ID	VARCHAR2 (50 BYTE)
HG_VERSION	VARCHAR2 (10 BYTE)
GENE_NAME	VARCHAR2 (50 BYTE)
ENTREZ_ID	VARCHAR2 (50 BYTE)

Table 5.2: TranSMART VARIANT\_SUBJECT\_DETAIL table

### 5.1.5 SeqWare

SeqWare is a Key Value implementation based on HBase designed by [OMN10]. As shown in Figure 5.4, SeqWare uses chromosome identity and SNP position as the key. The value consists of Families (column groups) and Qualifiers (columns in a group). Families and Qualifiers are flexible and online changeable.

The HBase database is a generic Key Value, column oriented database that pairs well with the inherent sparse matrix nature of variant annotations. The primary table stores multiple genomes worth of generic features, variants, coverages, and variant consequences using genomic location within a particular reference genome as the key. Each genome is represented by a particular column family label (such as variant:genome7). For locations with more than one called variant the HBase timestamp is used to distinguish each. Secondary indexing is accomplished using a secondary table per genome indexed. The key is the tag being indexed plus the ID of the object of interest, the value is the row key for the original table. This makes lookup by secondary indexes, tags for example, possible without having to iterate over all contents of the primary table.

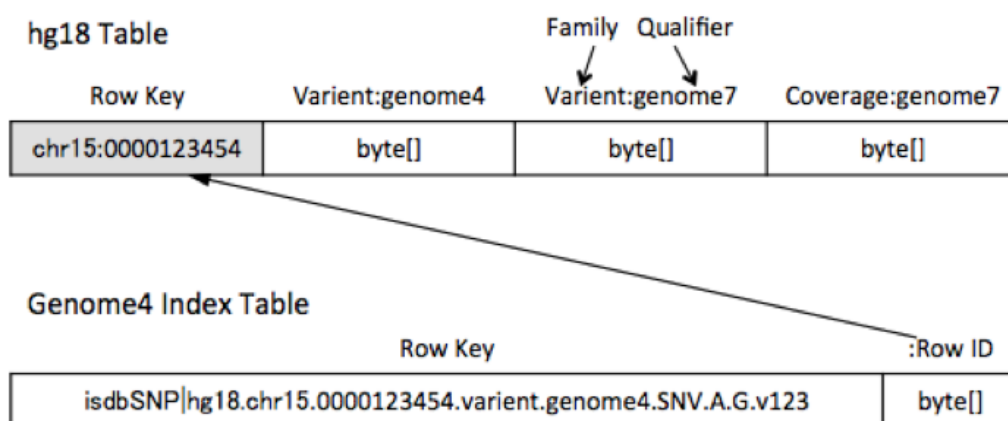


Figure 5.4: SeqWare Query Engine schema. The main table stores multiple genome information, such as generic features, variants and coverages. Each genome is represented by a particular Column Key (such as variant:genome6). Secondary indexing is implemented using a secondary table per genome indexed. The key is the column being indexed with the ID of the interest object. The value is represented by the row key in the main table. [OMN10]

## 5.2 SNP Key Value Model

Inspired from SeqWare, a new Key Value schema has been designed with a supplementary cluster index to overcome the disadvantages resulting from secondary index and provide fast query on all indexed properties, as shown in Figure 5.5. There are three tables Position, Subject and Cross-study, as designed in generic Key Value model. An extra Family 'info' is introduced to store general information for a SNP.

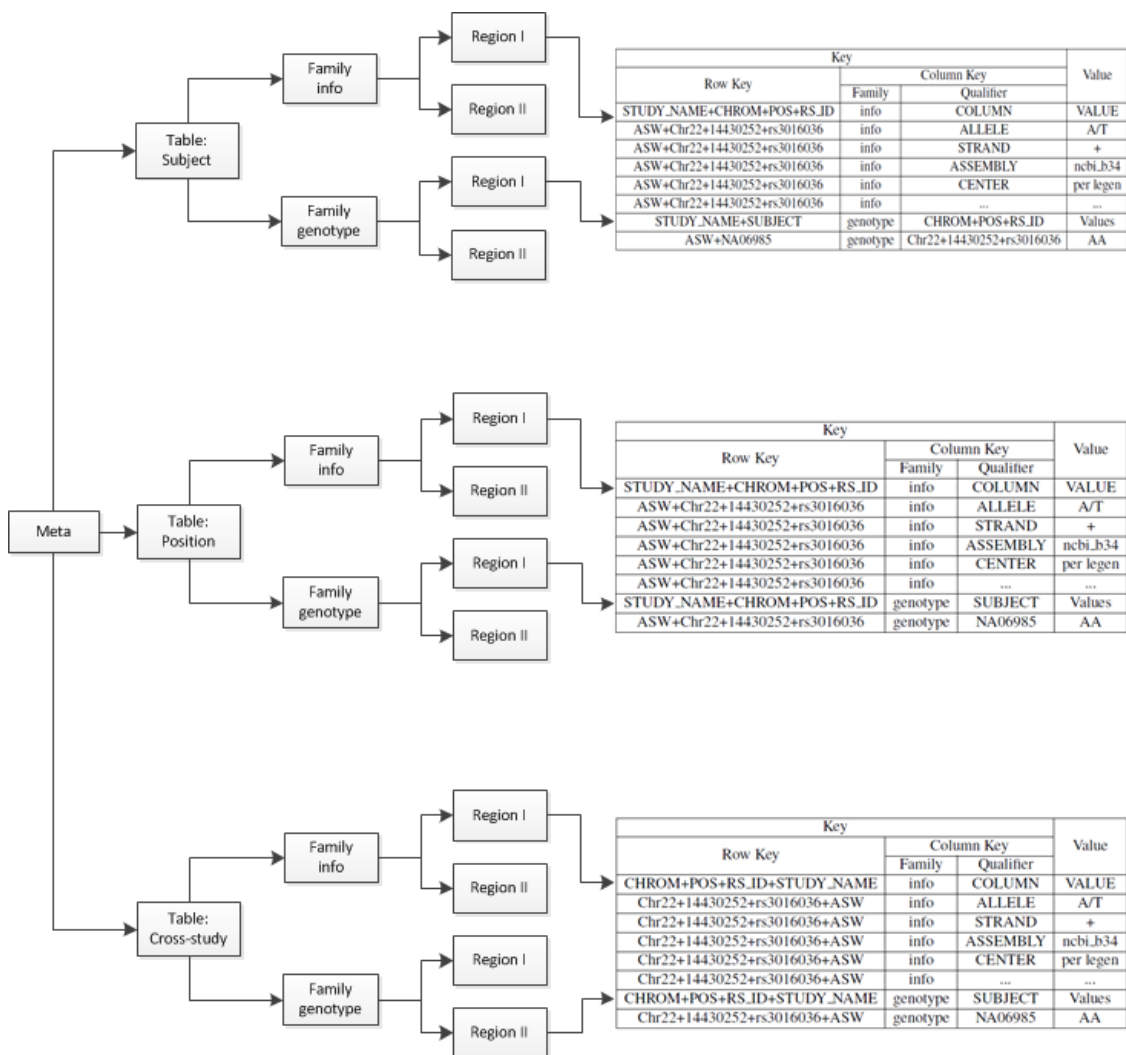


Figure 5.5: SNP Key Value data model.

In addition, a HapMap SNP data sample is added as an example and formatted using a simple relational database schema, as shown in Figure 5.6.

snp_id	allele	chrom	pos	strand	assembly	center
rs3016036	A/T	Chr22	14430252	+	ncbi_b34	per legen

ind_id	snp_id	study_name	Obs
NA06985	rs3016036	ASW	AA

← snp\_id

Figure 5.6: HapMap SNP data sample. There are two tables in this schema. The top table stores general SNP information. The bottom table stores sample and observation data.

### 5.2.1 Subject Table

The Subject table, as shown in Table 5.3, contains two Families, Family info and position. Dataset and Subject are composed of the Row Key for Key Value pairs. Family info utilizes the same structure as that in the Position table. This table offers high-speed query by subject in a dataset.

#### Determining Row Key and Column Key of transcriptomic data

Following the generic data model, the STUDY\_NAME, SUBJECT are placed in the Row Key for Family genotype, as shown in table 5.3. If Family info follows the same rule, data in this Family will be repeated. The number of duplication is subject number in this study. To avoid this large duplication, the Row Key for Family info is the same as the structure in the Position table.

#### Optimizing the Row Key to speed up data location

In order to speed up queries over subject in chromosome Families, the STUDY\_NAME and SUBJECT are placed in the Row Key to identify subjects. The order of the two fields in the Row Key also has a very significant effect on the query performance. Typical user cases may start with a gene information query of several subjects in a study. By placing the Dataset before Subject, the composite key becomes two indices on the Dataset and

Subject respectively. When multiple identical subjects are retrieved, this Row Key can precisely locate all HFiles expected by using this Row Key to compare to the Start keys of an HFile.

### Optimizing the Column Key to increase cache hit rate

One way to further increase cache hit rate is to design a new structure by classifying data in different chromosomes under Column Key. The Family divides different chromosomes into different HFiles and the Qualifier (POS) orders positions lexicographically. If queries by subject are required, the user requests data on several subjects that contain data only in one chromosome, but the query returns millions of records. Such large numbers of gene records can be loaded with only a limited number of disk loading operations through the corresponding Column Keys.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
STUDY_NAME+CHROM+POS+RS_ID	info	COLUMN	VALUE
ASW+Chr22+14430252+rs3016036	info	ALLELE	A/T
ASW+Chr22+14430252+rs3016036	info	STRAND	+
ASW+Chr22+14430252+rs3016036	info	ASSEMBLY	ncbi_b34
ASW+Chr22+14430252+rs3016036	info	CENTER	per legen
ASW+Chr22+14430252+rs3016036	info	...	...
STUDY_NAME+SUBJECT	genotype	CHROM+POS+RS_ID	Values
ASW+NA06985	genotype	Chr22+14430252+rs3016036	AA

Table 5.3: TranSMART SNP Subject table

### 5.2.2 Position Table

The Subject table only provides faster queries over subject within one study. However, if queries by genetic position are required, the Subject table queries may not be performed

as fast as the queries by subject. The Position table is designed to provide fast query by position.

In the Position table, as shown in Table 5.4, the Row Key is a composite key including fixed length STUDY\_NAME, CHROM, POS and RS\_ID; while the value consists of two families. Family info contains the properties in table DE\_RC\_SNP\_INFO; Family subject contains the Values, such as genotypes or phenotypes, for all the subjects in this dataset. This table provides efficient query for SNP position in a dataset.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
STUDY_NAME+CHROM+POS+RS_ID	info	COLUMN	VALUE
ASW+Chr22+14430252+rs3016036	info	ALLELE	A/T
ASW+Chr22+14430252+rs3016036	info	STRAND	+
ASW+Chr22+14430252+rs3016036	info	ASSEMBLY	ncbi_b34
ASW+Chr22+14430252+rs3016036	info	CENTER	per legen
ASW+Chr22+14430252+rs3016036	info	...	...
STUDY_NAME+CHROM+POS+RS_ID	genotype	SUBJECT	Values
ASW+Chr22+14430252+rs3016036	genotype	NA06985	AA

Table 5.4: TranSMART SNP position table

### Determining Row Key and Column Key of transcriptomic data

In order to speed up queries by reference position, the STUDY\_NAME, CHROM, POS and RS\_ID are placed in the Row Key, as shown in table 5.4, because these four fields are the minimum information that can identify a row in a VCF or HapMap file. The three fields STUDY\_NAME, CHROM and POS can locate data by reference position. But different SNPs may be mapped to the same position even within one study. Thus, RS\_ID is added to the Row Key.



### **Optimizing the Row Key to speed up data location**

The order of the four fields in the Row Key also affect on the query performance. Typical user cases always begin with a gene information query of several positions. By placing the STUDY\_NAME before the CHROM and POS (STUDY\_NAME + CHROM + POS), the composite key becomes three indices on the dataset, chromosome and position respectively. When multiple identical position in a dataset are retrieved, STUDY\_NAME + CHROM + POS can precisely locate all HFiles expected by using STUDY\_NAME + CHROM + POS to compare to the Start keys of an HFile. After POS, RS\_ID is placed to make this Row Key unique.

### **Optimizing the Column Key to increase cache hit rate**

One way to further increase cache hit rate is to design a new structure by classifying different types of data under Column Key. The Family divides different types of data (info and genotype) into different HFiles and the Qualifier orders the general (info) and subject values (genotype) lexicographically. In most use cases, the user requests data on several positions that contain only genotype values, but the query returns millions of records. Such large numbers of genotype records can be loaded with only a limited number of disk loading operations through the corresponding Column Keys. The HFiles storing the data in these positions will be loaded into BigTable caches. Thus, only retrieving the first record causes a significant delay when loading data from disk to memory, while all other records in the same HFile are fetched directly from memory caches.

### **5.2.3 Cross-study Table**

This Position table only provides faster queries over SNP within one study. However, if queries over multiple datasets are required, these queries may not be performed as fast as

the queries by position within one study. The Cross-study table is designed to provide fast query by position across studies.

In the Cross-study table, as shown in Table 5.5, the Row Key is a composite key including fixed length STUDY\_NAME, CHROM, POS and RS\_ID; while the value consists of two families. Family info contains the properties in table DE\_RC\_SNP\_INFO; Family genotype contains the Values for all the subjects in this study. This table provides efficient query by position across datasets.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
CHROM+POS+RS_ID+STUDY_NAME	info	COLUMN	VALUE
Chr22+14430252+rs3016036+ASW	info	ALLELE	A/T
Chr22+14430252+rs3016036+ASW	info	STRAND	+
Chr22+14430252+rs3016036+ASW	info	ASSEMBLY	ncbi_b34
Chr22+14430252+rs3016036+ASW	info	CENTER	per legen
Chr22+14430252+rs3016036+ASW	info	...	...
CHROM+POS+RS_ID+STUDY_NAME	genotype	SUBJECT	Values
Chr22+14430252+rs3016036+ASW	genotype	NA06985	AA

Table 5.5: TranSMART SNP Cross-study table

### Determining Row Key and Column Key of transcriptomic data

In order to speed up queries by reference position across datasets, the STUDY\_NAME, CHROM, POS and RS\_ID are placed in the Row Key, as shown in table 5.4, because these four fields are the minimum information that can identify a row in a VCF or HapMap file. The three fields STUDY\_NAME, CHROM and POS can locate data by reference position. But different SNPs may be mapped to the same position even in one study. Thus, RS\_ID is added to the Row Key.

### Optimizing the Row Key to speed up data location

The order of the four fields in the Row Key affect on the query performance. Typical user cases always begin with a gene information query by position across multiple datasets. By placing the CHROM and POS before STUDY\_NAME (CHROM + POS + STUDY\_NAME), the composite key becomes three indices on the chromosome, position and dataset respectively. When multiple identical positions across datasets are retrieved, CHROM + POS + STUDY\_NAME can precisely locate all HFiles expected by using CHROM + POS to compare to the Start keys of an HFile. After STUDY\_NAME, RS\_ID is placed to make this Row Key unique.

## 5.3 Performance Evaluation

An experiment was performed using three datasets (first three datasets from the 2010-05 phase III consensus datasets of HapMap project [Con07]: African ancestry in Southwest USA (ASW), 87 samples and 119,487 SNPs; Utah residents with Northern and Western European ancestry from the CEPH collection (CEU), 165 samples and 119,487 SNPs; Han Chinese in Beijing, China (CHB), 137 samples and 317,642 SNPs) from HapMap project to test this new Key Value data schema on a cloud computing test-bed, OpenStack, installed at Imperial College London. In this experiment a database with the dbSNP schema that NCBI was used to store SNP data and mimic general procedures for clinical decision making. The data are formatted and uploaded into MySQL Cluster and HBase instances respectively. Each database cluster is comprised of 4 virtual machines, each with 8 CPU cores and 8 GB memory. Each VM used a 100 GB disk. HBase was used to implement the Key Value model, while MySQL Cluster implements the relational model. Each database was configured as follows:

- HBase (Version 0.96.0 on Hadoop 1.0.3): One master server node and three slave

nodes with HBase configured in fully distributed mode. The master server was configured as a HMaster and a HRegionServer, while each slave node was configured as a HRegionServer. To implement the relational model shown in Figure 5.6, two tables are created (SNPT, upper;OBT, lower). In SNPT the `snp_id` is unique and placed in the Row Key, all other columns are placed in the Column Key. Column `chrom` and Column `pos` are indexed by B-tree. In OBT there is no unique id, so an extra id is added as Row Key. OBT indexes follows Figure 3.1, where Column position is Column `snp_id`.

- MySQL Cluster (Version 5.6.11-ndb-7.3.2): Four VMs, with one as a manager node and three data nodes. The manager node consists of a MGM, a MySQL and a NDB using one VM. Each data node consisted of a NDB. Disk-based InnoDB storage engine was used, where indexed columns and indices are stored in memory and non-indexed columns are stored in disks. The relational model implementation is the same as that in HBase. Two data copies were used to guarantee its data consistency.

### 5.3.1 Query by subject

The first step of the experiments is to find significant differential SNPs. With clinical information in a study, such as cohorts and clinical measurement statistics, all available SNPs of the specified subjects with typical features are loaded for further tests to find the expected SNPs.

MySQL Cluster used observation and SNP tables. There are two types of queries frequently used. One is used for range query and the other is random read query. The first query is usually applied on the main cohorts (almost all subject) at the beginning of a study to roughly find most relevant SNPs. When multiple cohorts are required, the traditional method generates corresponding random read queries, which are much slower than

range queries. Currently this tradition method becomes unacceptable slow due to the fast increasing NGS data amount. Thus, it is much more efficient to merge these queries respectively for each cohort into one big query using range query. The random query is only used for small queries that can be finished within a few seconds. In order to test the random read performance two subjects from each dataset are randomly chosen. The number of SNPs in datasets ASW and CEU is 119487 and the number of CHB is 317642. MySQL Cluster used the two statements: range scan query, as shown in Figure 5.7; random read query, as shown in Figure 5.8.

```

SELECT
  *
FROM
  snpdata.Obs o, snpdata.SNP s
WHERE
  o.snp_id = s.snp_id
AND
  o.dataset = 'dataset'
AND
  o.ind_id >= 'NA00000'
AND
  o.ind_id <= 'NA99999';

```

Figure 5.7: Range scan by position within one study.

```

SELECT
  *
FROM
  snpdata.Obs o, snpdata.SNP s
WHERE
  o.snp_id = s.snp_id
AND
  o.dataset = 'ASW'
AND
  o.ind_id IN ('NA00000',..., 'NA99999');

```

Figure 5.8: Random read by position within one study.

For HBase, the subject table is the most suitable one for this query. Two correspond queries are designed for range scan and random read.

range scan query: Function Scan using starting row ASW+NA00000 and ending row ASW+NA99999.

random read query: Function Get with List<Get>, each Get represents a subject such as

NA00000,..., NA99999.

When querying by subject within a single dataset, as shown in Figure 5.9, the Key Value model on HBase shows an average 269.39 times faster than the relational model on HBase. The Key Value model on HBase outperforms the relational model on HBase by 351.33 times in CHB dataset, while in datasets ASW and CEU, the Key Value model respectively conducts 215.15 and 241.68 times faster than the relational model on HBase Cluster. The Key Value model on HBase shows about 4 or more times faster data retrieval time than the relational model on MySQL. Particularly in ASW, The Key Value model on HBase outperforms the relational model on MySQL by 9.12 times. While in datasets CEU and CHB, the Key Value model respectively conducts 5.12 and 4.51 times faster than the relational model on MySQL Cluster.

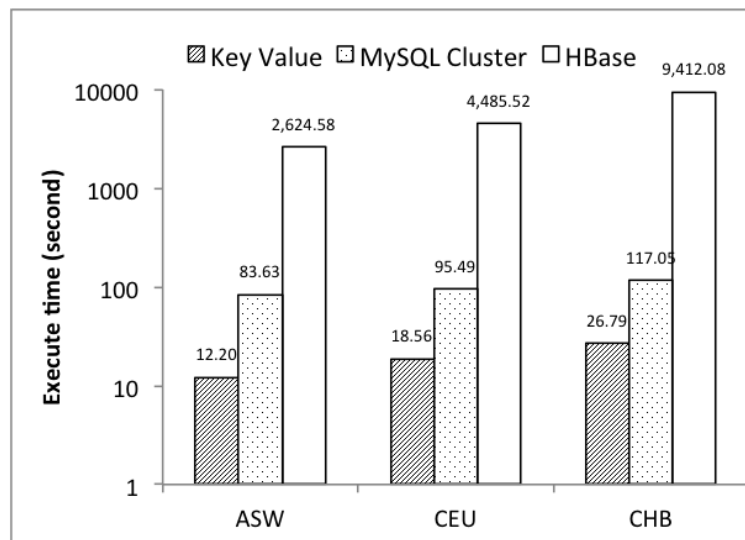


Figure 5.9: Query by subject. The Key Value bar represents the Key Value model on HBase. All the other bars represents the relational model on a specific database.

The response time of one subject queries over 119,487 SNPs the Key Value data model on HBase are 148.25 times faster than the relational model on HBase on average, as shown in Figure 5.10, while the queries over 317,462 SNPs using the Key Value model outperformed the relational model on HBase by 99.20 times. The response time of 119,487 SNPs queries with the Key Value model are 2.61 times faster than the relational one on

MySQL Cluster on average, while the queries over 317,462 SNPs using the Key Value outperformed the relation one on MySQL Cluster by 1.57 times.

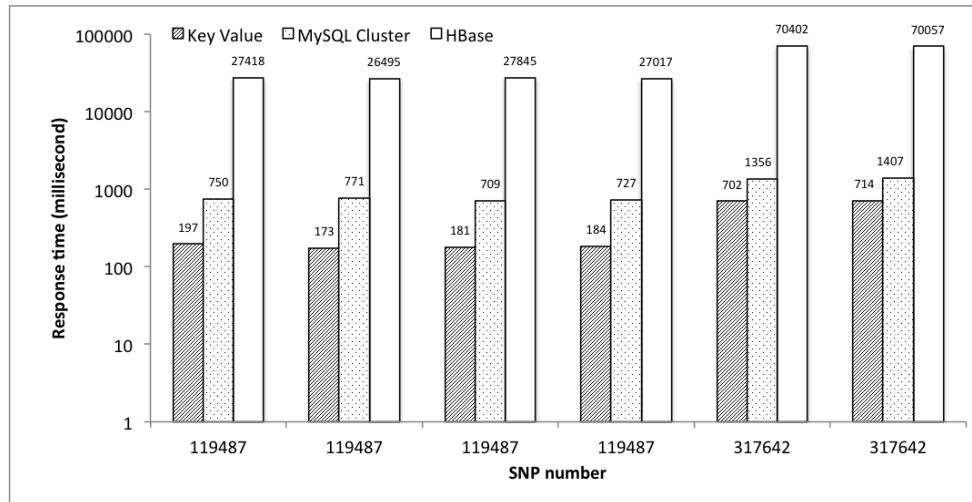


Figure 5.10: Random read 1 subject. The Key Value bar represents the Key Value model on HBase. All the other bars represents the relational model on a specific database.

### 5.3.2 Query by SNP position

After differential expressed SNPs are calculated, the top hundred SNPs are usually tested by other datasets to further reduce the number of these SNPs. If SNPs associated with certain diseases are found, queries over a several SNPs across datasets are generated to detect whether a patient is vulnerable to certain diseases based on his genotypes of the SNPs. For example, numerous SNPs have each been associated with slightly increased risk for type-2 diabetes and improve the probability of predicting whether an individual is vulnerable to type-2 diabetes based on the traditional clinical characteristics, such as age, sex and weight [VDW<sup>+</sup>08]. 10 such SNPs (rs4402960, rs775840, rs10811661, rs9300039, rs8050136, rs1801282, rs13266634, rs1111875, rs7903146, rs5219) have been found in multiple studies and meta-analyses [SMB<sup>+</sup>07]. If you have the highest risk genotype for all 10 of these SNPs, you are estimated to be at double the risk for type-2 diabetes compared to the average person, whereas if you have the lowest risk genotype for every one

of these 10 SNPs, you are estimated to be at half the risk. Another tens of SNPs relevant to type-2 diabetes have been reported [CL12].

Therefore, a test case to query 100 discrete SNPs by position is generated and applied on each dataset and the whole database. This fully random read query cannot be optimised by any range scan. MySQL Cluster used the two statements: query over one study, as shown in Figure 5.11; query over multiple trial, as shown in Figure 5.12.

```
SELECT
  *
FROM
  snpdata.Obs o, snpdata.SNP s
WHERE
  o.snp_id = s.snp_id
AND
  o.dataset = 'dataset'
AND
  s.chrom IN ('chr1')
AND
  s.pos IN ('23211432', '78456956', ...);
```

Figure 5.11: Query over one study.

```
SELECT
  *
FROM
  snpdata.Obs o, snpdata.SNP s
WHERE
  o.snp_id = s.snp_id
AND
  s.chrom IN ('chr1')
AND
  s.pos IN ('23211432', '78456956', ...);
```

Figure 5.12: Query over multiple studies.

For HBase, Position and Cross-study tables used Function Get and List<Get> for query over a dataset and the whole database respectively. Each Get object represents a SNP position such as 23211432 and 78456956.

While querying 100 discrete SNP by position, the Key Value model on HBase performs an average 15.40 times faster than the relational model on HBase. But the relational model on MySQL Cluster shows 1.84 times faster than the Key Value HBase on average, as shown



in Figure 5.13, due to the slow HBase Random Read. However, MySQL Cluster does not scale well when concurrent queries across studies are considered, as shown in Figure 5.14. The Key Value based data model demonstrates an average 4.66 times of increase in query processing speed compared to the relational model implemented on MySQL Cluster. In 6/10 of cases, the Key Value data model is more than 5 times faster than the relational model on MySQL Cluster. In the worst case, when using only 10 threads, the Key Value data model is more than 3.07 times faster than relational model on MySQL Cluster. Compared to the relational model on HBase, the Key Value one performs 105.40 times faster on average. Especially, in 6/10 of cases, the Key Value model is more than 100 times faster than the relational one on HBase.

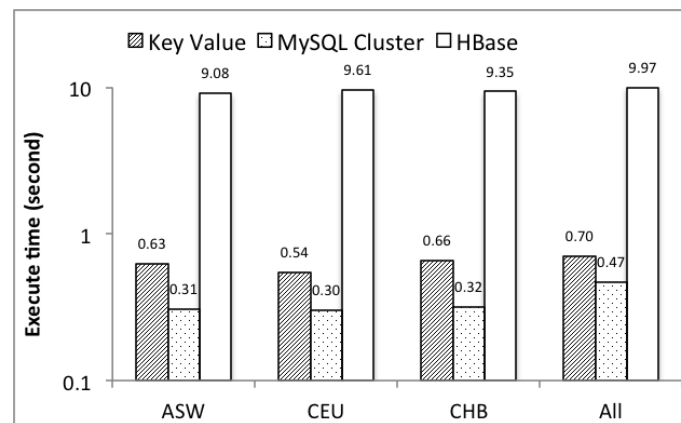


Figure 5.13: Random read 100 SNPs. The Key Value bar represents the Key Value model on HBase. The Key Value tests in left three groups use the Position table and the one in the last group uses the Cross-study table. All the other bars represents the relational model on a specific database.

## 5.4 Integration into tranSMART

This implementation is based on HBase 0.96.0. As shown in the Figure 2.11, the study and subject information is retrieved from a SQL database before a variant data query. Three types of data queries are created corresponding to the three complementary tables. Figure 5.15 shows the variant data query by subject via the subject table using the Random Read

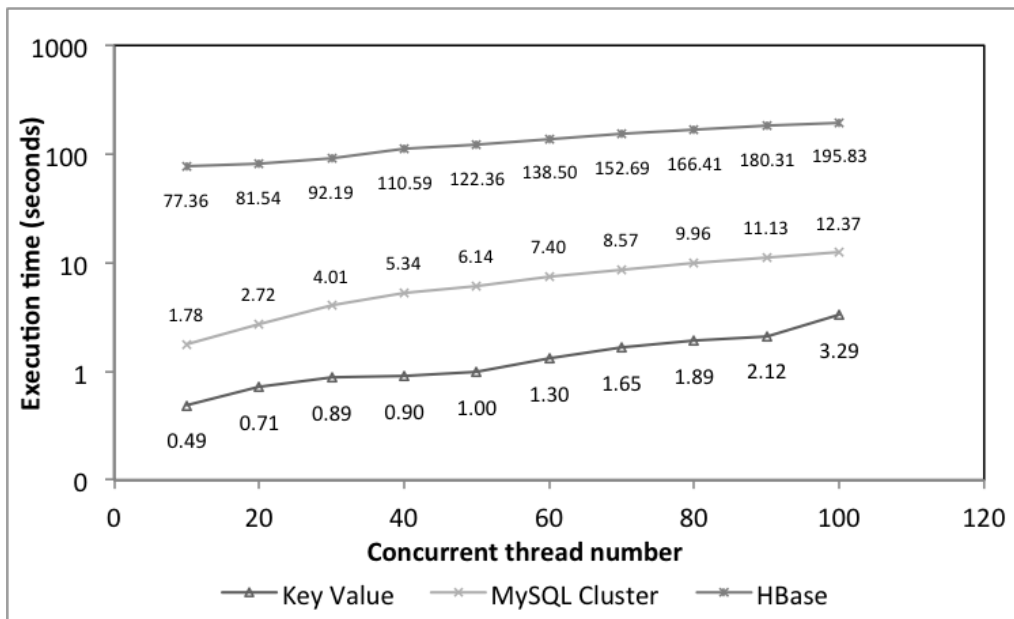


Figure 5.14: Concurrent random read 100 SNPs. The Key Value curve represents the Key Value model on HBase using the Cross-study table. All the other curves represents the relational model on a specific database.

method; Figure 5.16 indicates the data query via the position table using the sequential Scan method; Figure 5.17 describes the query method via the cross-study table using the Scan method. The cacheSize used in the last two Figures may be changed according to the physical environments and the data scale.

```

/*
 * retrieve data by subject using the Random Read function
 * @parameter trialName: the name of the trial(study)
 * @parameter patientList: the list of patients
 */
getVarRecordsBySubject(trialName, patientList) {
    Result[] results = variant-subjectTable.get(trialName + patientList);
    for (int i = 0; i < results.length; i++)
        for (Cell kvPair : results[i].rawCells()) {
            results.add(kvPair);
        }
}

```

Figure 5.15: The pseudocode of variant data query by subject.

```

/*
 * retrieve data by position within a trial(study) using the Scan function
 * @parameter trialName: the name of the trial(study)
 * @parameter startPos: the genetic starting position
 * @parameter endPos: the genetic ending position
 * @parameter cacheSize: the buffer size of the Scan function
 */
scanVariantTableByPos(trialName, startPos, endPos, cacheSize) {
    Scan s = new Scan();
    // set cache to tuning the performance
    s.setCacheBlocks(true);
    s.setCaching(cacheSize);
    s.setStartRow(trialName + startPos);
    s.setStopRow(trialName + endPos);
    List results;
    long count = 0;
    ResultScanner scanner = variant-posTable.getScanner(s);
    for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
        for (Cell kvPair : rr.rawCells()) {
            results.add(kvPair);
        }
    }
    scanner.close();
    return results;
}

```

Figure 5.16: The pseudocode of variant data query by position within a study.

```

/*
 * retrieve data by position within a trial(study) using the Scan function
 * @parameter startPos: the genetic starting position
 * @parameter endPos: the genetic ending position
 * @parameter cacheSize: the buffer size of the Scan function
 */
scanVariantTableByPosAcross(startPos, endPos, cacheSize) {
    Scan s = new Scan();
    // set cache to tuning the performance
    s.setCacheBlocks(true);
    s.setCaching(cacheSize);
    s.setStartRow(startPos);
    s.setStopRow(endPos);
    List results;
    long count = 0;
    ResultScanner scanner = variant-acrossTable.getScanner(s);
    for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
        for (Cell kvPair : rr.rawCells()) {
            results.add(kvPair);
        }
    }
    scanner.close();
    return results;
}

```

Figure 5.17: The pseudocode of variant data query by position across studies.

### 5.4.1 Benchmark

An experiment was performed with the same datasets (African ancestry in Southwest USA (ASW), 87 samples and 119,487 SNPs; Utah residents with Northern and Western European ancestry from the CEPH collection (CEU), 165 samples and 119,487 SNPs;

Han Chinese in Beijing, China (CHB), 137 samples and 317,642 SNPs) used in the performance evaluation section to test the time consuming data export function. Each dataset will be downloaded through tranSMART data export function.

### 5.4.2 Database configuration

- HBase (Version 0.96.0 on Hadoop 1.0.3): One master server VM (4 CPU cores, 4 GB memory) and three slave VMs (3x4 CPU cores, 3x4 GB memory) with HBase configured in fully distributed mode. The master server was configured as a HMaster and a HRegionServer, while each slave node was configured as a HRegionServer. The Subject table in the Key Value model is chosen to export data by subject.
- PostgreSQL (Version 9.4.1): One VM was used with 16 CPU cores and 16 GB memory, on which a standard tranSMART configuration was applied. This is the original database in tranSMART.

### 5.4.3 Results

In Figure 5.18 the HBase implementation outperformed the PostgreSQL by average 5.64 times speedup. In the ASW case, HBase performed 6.87 times faster than PostgreSQL, whilst in CEU and CHB cases HBase showed 4.99 and 5.06-fold speedup respectively.

## 5.5 Discussion

When using Subject table to perform queries by subject, SNP information are not stored together with genotypes sharing the same Row Key. This may lead to extra random read operations when subject number in a dataset is very few. The worst case is only one

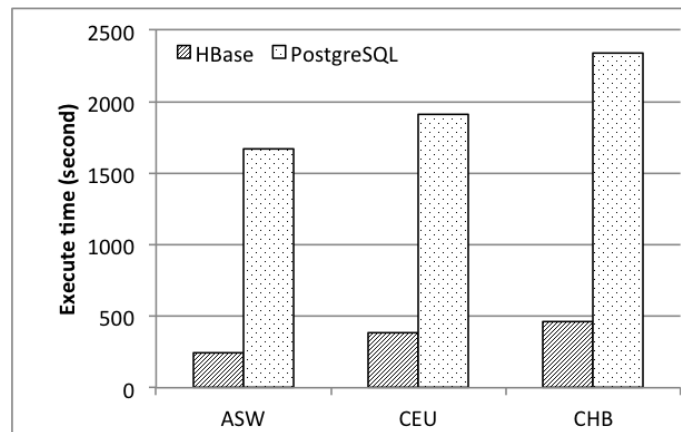


Figure 5.18: Performance evaluation on tranSMART.

subject in a dataset. Then the subject data loading time may be doubled due to the extra search for SNP information.

The stability of the relational model in MySQL Cluster is higher than the Key Value model in HBase. The main reason is that the short query in HBase is easy to be disturbed by the periodical HBase system maintenance and optimization operations, such as data consistence checking, Region splitting or Region merging. Such operations can delay a query by a few hundreds of milliseconds. Compared to the usual short query time (only very few seconds), this delay could lead to a very large deviation. This is also the reason why longer queries are more stable.

In Section Integration into tranSMART, the throughput of the Key Value implementation is much lower than the results in Section Performance evaluation. The most important one is the download file cache. Due to the current slow database export function, the current tranSMART provides an asynchronous export function, which extracts data from the database and saved in the local file system first. When data is fully extracted, users can download the file. This disk writing operation slows down the high throughput of the Key Value implementation and will be improved in the future.

## 5.6 Conclusion

The SNP Key Value model is aimed at solving the performance issues faced in translational research data storage databases, due to the increasing volume of data being produced. This chapter indicates a Key Value pair model leads to an increase in performance when querying genotype or phenotype biological data, over the relational model currently implemented in tranSMART. The Key Value model on HBase outperforms the relational model on HBase and MySQL Cluster by up to 9.12-fold and 351.33-fold speedup respectively. When integrated into tranSMART this Key Value implementation performs up to 6.87 times faster than the current tranSMART.

# Chapter 6

## Raw Sequencing Key Value Model

### 6.1 Introduction

A sequence alignment [MM01] is a method of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns. If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps can be interpreted as indels (insertion or deletion mutations) [SSH<sup>+</sup>01].

Very short or very similar sequences can be aligned manually. However, most alignment cases require the alignment of lengthy, highly variable or numerous sequences that is extremely difficult to be aligned by human effort. Thus, alignment algorithms have been designed to produce high-quality sequence alignments automatically and occasionally manual adjustments of the final results are performed to reflect patterns that are difficult to represent algorithmically. Generally, computational methods to sequence alignment

fall into two categories: global alignments [HC03] and local alignments [AGM<sup>+</sup>90]. Calculating a global alignment is a form of global optimization that forces the alignment span the entire length of all query sequences. By contrast, local alignments identify regions of similarity within long sequences that are often widely divergent overall. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity. A variety of computational algorithms have been applied to the sequence alignment. Some of these algorithms are slow but formally correct, such as dynamic programming, while the other algorithms are efficient but can not guarantee to find best matches, such as heuristic algorithms or probabilistic methods designed for large-scale database search.

Alignments are commonly represented both in graphical and text format, for example, GenomeBrowser and SAM. In most sequence alignment representations, sequences are written in rows arranged so that aligned residues appear in successive columns. In text formats, aligned columns containing identical or similar characters are indicated with a system of conservation symbols. Many sequence visualization programs also use color to display information about the properties of the individual sequence elements. In DNA and RNA sequences, each nucleotide is assigned its own color.

Sequence alignments can be stored in a wide variety of text-based file formats, many of which were originally developed in conjunction with a specific alignment program or implementation. Most web-based tools allow a limited number of input and output formats, such as FASTA format [CSK<sup>+</sup>03] and GeneBank format [FLW03] and the output is not easily editable. Several conversion programs that provide graphical and/or command line interfaces are available, such as ReadSeq [Gil03] and EMBOSS [RLB<sup>+</sup>00]. There are also several programming packages which provide this conversion functionality, such as BioPerl [SBB<sup>+</sup>02] and BioRuby [GPN<sup>+</sup>10].

Global alignments, which attempt to align every residue in every sequence, are most use-



ful when the sequences in the query set are similar and of roughly equal length. A general global alignment technique is usually based on dynamic programming [BBBB95]. Local alignments are much better for different sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. Hybrid methods, known as semi-global methods, attempt to find the best possible alignment that includes the start and end of one or the other sequence. Pairwise sequence alignment methods are used to find the best-matching alignments of two query sequences. Pairwise alignments can only be used between two sequences at a time, but are efficient to calculate for methods that do not require extreme precision, such as searching a database for sequences with high similarity to a query. Multiple sequence alignment is an extension of pairwise alignment to process more than two sequences at a time. Multiple alignment methods try to align all of the sequences in a given query set. Multiple alignments are widely used in identifying conserved sequence regions across a group of sequences. Multiple sequence alignments are computationally difficult to produce and most formulations of the problem lead to NP-complete combinatorial optimization problems.

Sequenced RNA, such as expressed sequence tags and full-length mRNAs, can be aligned to a sequenced genome to find where there are genes and get information about alternative splicing and RNA editing. Sequence alignment is also a part of genome assembly, where sequences are aligned to find overlap so that contigs can be formed. Another use is SNP analysis, where sequences from different individuals are aligned to find single basepairs that are often different in a population.

RNA-seq has been used for profiling of transcriptomes in many areas of biology, including studies in medicine and pharmacy. The purpose of this chapter is to improve clinical decision making through the discovery of differentially expressed genes across patient and control cases in a cohort. RNA-seq data processed by the protocol [AMC<sup>+</sup>13] is a classic example of the molecular profiling based patient stratification function with sequence data.

High-throughput transcriptomic data generated by NGS technologies [LLB<sup>+</sup>01, VAM<sup>+</sup>01, RKP<sup>+</sup>96] is the most popular data type currently used in translational studies. With its unprecedented throughput, scalability, and speed, NGS enables researchers to study biological systems at a level never before possible. For the needs of various collaborative translational research projects, transSMART is planning to support NGS alignment data for translational studies. Anticipating the requirement to store and analyze NGS data, where the volume of data being produced will be in the terabyte (TB) range, the current performance exhibited by BAM is unacceptably poor.

A typical query involves searching for alignments overlapping a specific region for a gene or a trait to count its expression level [RIST<sup>+</sup>12, RSS<sup>+</sup>12] and perform data analysis. The current transSMART uses BAM files to store alignment data. Once retrieved the data is passed to analytical tools, such as Samtools [LHW<sup>+</sup>09], UCSC Genome Browser [KHK11] and custom R workflows for further analysis or visualization.

## 6.2 Background

### 6.2.1 SAM

SAM format [SS11] is a TAB-delimited text format comprised of an optional header section and an alignment section. Header lines begin with @, but alignment lines do not. Each alignment line has 11 mandatory fields for crucial alignment information such as mapping position, and variable number of optional fields for flexible or aligner specific information. An ever increasing sequence data amount leads to a significant scalability problem of SAM.

Each header line starts with @ followed by a two-character code indicating a record type (HTC). In the header, every line is TAB-delimited. Except the lines beginning with @CO,

each data field uses a format TAG:VALUE in which TAG is a two-letter code that defines the content and the format of VALUE.

In the SAM format, each alignment data record shows the linear alignment of a segment. There are 11 required fields in each line, as shown below.

- QNAME: Query template NAME
- FLAG: bitwise FLAG
- RNAME: Reference sequence NAME
- POS: 1-based leftmost mapping POSition
- MAPQ: MAPping Quality
- CIGAR: CIGAR string
- RNEXT: Ref. name of the next read
- PNEXT: Position of the next read
- TLEN: observed Template LENgth
- SEQ: segment SEQuence
- QUAL: ASCII of Phred-scaled base QUALity+33

If the information in a field is unavailable, the corresponding value can be 0 or \*. Optional fields follow the TAG:TYPE:VALUE format where TAG is a two-character string that starts with a letter and ends with a letter or a number. Each TAG can only appear once in one alignment line. More details about SAM can be found at [SS11].

CIGAR is the most important field that can be used to describe the details of mapping.

- M: alignment match (can be a sequence match or mismatch)

- I: insertion to the reference
- D: deletion from the reference
- N: skipped region from the reference; For mRNA-to-genome alignment, an N operation represents an intron. For other types of alignments, the interpretation of N is not defined.
- S: soft clipping (clipped sequences present in SEQ); S may only have H operations between them and the ends of the CIGAR string.
- H: hard clipping (clipped sequences NOT present in SEQ); H can only be present as the first and/or last operation.
- P: padding (silent deletion from padded reference)
- =: sequence match
- X: sequence mismatch

There are two methods to calculate the mapped length in a reference sequence: sum of lengths of the M, D, N, = and X is the length without padding; sum of lengths of the M, D, N, =, X and P is the padding length.

This is a data sample from 1000 Genomes project, shown in Figure 6.1.

## 6.2.2 BAM

BAM file [BGQ<sup>+</sup>11] is compressed SAM file implemented on top of the standard gzip file format. The UCSC (University of California, Santa Cruz) binning indexing [KSF<sup>+</sup>02] applied on BAM aims to achieve fast retrieval of alignments overlapping a specified region without scanning the whole alignments. In this index, each bin represents a contiguous

```
@HD VN:1.0 SO:coordinate
@SQ SN:1 LN:249250621 M5:1b22b98cdeb4a9304cb5d48026a85128 UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/
phase2_reference_assembly_sequence/hs37
d5.fa.gz AS:NCBI37 SP:Human
@SQ SN:2 LN:243199373 M5:a0d9851da00400dec1098a9255ac712e UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/
phase2_reference_assembly_sequence/hs37
d5.fa.gz AS:NCBI37 SP:Human
@SQ SN:3 LN:198022430 M5:fdfd811849cc2f4deb929bb925902e5 UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/
phase2_reference_assembly_sequence/hs37
d5.fa.gz AS:NCBI37 SP:Human
@RG ID:SRR741384 LB:IWG_IND-TG.HG00097-4_1pA SM:HG00097 PI:297 CN:BCM PL:ILLUMINA DS:SRP001294
@RG ID:SRR741385 LB:IWG_IND-TG.HG00097-4_1pA SM:HG00097 PI:297 CN:BCM PL:ILLUMINA DS:SRP001294
@PG ID:bwa_index PN:bwa VN:0.5.9-r16 CL:bwa index -a bwtsv $reference_fasta
SRR741384.47119843 163 20 59993 29 68532M15 = 60191 263
AAATATTGAGTAATACTCCCACTTTGAACAGAATTTTAAAGAGAAAAACTGAAAGTTAATAGAGAGGTGACTCAGATCCAGAGGTGGAAGAGGAAGAA .JJDDFBGFKBAHFDJEEI;M9FDHFJGHEHH
GHHHFF7BMHHEGCG4H;GJH0B>AI;MFHJJIH0CGACNQLP757HQF<;6@H6@A7<BB@JK## XC:i:100 MD:Z:0N0N0N0N0N0N0N24 RG:Z:SRR741384 AM:i:29
NH:i:8 SM:i:29 XN:i:8 MQ:i:29 XT:A:M
BQ:Z:#####S#####5#####BF##
SRR741385.13163696 163 20 59993 29 7594M = 60122 228
TAGAGAGGTGACTCAGATCCAGAGGTGGAAGGAGGAAGCTTGGAAACCTATAGAGTTGCTGAGTGCCAGGACCAGATCCTGGCCCTAAACAGGTGGTA .IHEBBAD:8BG>BFF@BG7GEGEC7
CHGLDARBEJIDJICIFILGKJIKGKFKFIJJIJFJGJLJFDGKSFHQKJKGK->FLGSSQIQIQC75 MD:Z:0N0N0N0N0N0N0N86 RG:Z:SRR741385 AM:i:29 NM:i:8
SM:i:29 XN:i:8 MQ:i:29 XT:A:M
BQ:Z:#####E#####
SRR741385.28094023 99 20 59993 29 50551M = 60199 306
CCCACTTTGAACAGAATTTTAAAGAGAAAAACTGAAAGTTAATAGAGAGGTGACTCAGATCCAGAGGTGGAAGGAGGAAGCTTGGAAACCTATAGAG .RRGGIGHGKHHHLLJHHHGLLJLJMLMMJ
KKGMMIHHMKMKMJLKFILLLKLLKMKKIKKIFJLJLJGDSKHHASHKSDSIIHIKKKQRCQF5 MD:Z:0N0N0N0N0N0N43 RG:Z:SRR741385 AM:i:29 NM:i:8 SM:i:29
XN:i:8 MQ:i:29 XT:A:M BQ:Z:#####W#####
SRR741385.76706450 99 20 59993 29 10584M75 = 60392 495
TAATAGAGAGGTGACTCAGATCCAGAGGTGGAAGGAGGAAGCTTGGAAACCTATAGAGTTGCTGAGTGCCAGGACCAGATCCTGGCCCTAACCAAGGTG .NSGFFHGHII;EEHKILJJKILLIHDNDHH
E>@IHHIILJLJIIJLJLEJLKLHND;OLGEKQFEPJR<6?DHHJSLIFAS55SRMD=;G##### XC:i:95 MD:Z:0N0N0N0N0N0N0N76 RG:Z:SRR741385 AM:i:29 NM:i:8
SM:i:29 XN:i:8 MQ:i:29 XT:A:M
BQ:Z:#####E#####D#####
SRR741385.72580568 121 20 59993 37 101M = 59993 0
GTGACTCAGATCCAGAGGTGGAAGGAGGAAGCTTGGAAACCTATAGAGTTGCTGAGTGCCAGGACCAGATCCTGGCCCTAAACAGGTGGTAAGGAAGG .SAC@??
SSCMRHLFNCKCKKDKKCIQ3HHLMLMIQHFIJIDGEGHKKHHPDAEELJHJLJLJGGGHGKJIHEFIHDDLEGFHCIIHIDHHBJIDERJ. X0:i:1 X1:i:0 MD:Z:0N0N0N0N0N0N0N93
RG:Z:SRR741385 AM:i:0 NM:i:8 SM:i:37 XN:i:8 XT:A:U
BQ:Z:E*KMIGFXT#####E
```

Figure 6.1: SAM data sample

genomic region. One bin is either fully contained in or non-overlapping with another bin. An alignment record is contained a bin which represents the smallest region containing the entire alignment. The binning index is represented by a R-tree. Each bin corresponds to an internal node in a R-tree [Gut84]. For example, bin A is a child of Bin B, if bin A is contained in B.

The first step to find the alignments overlapping a specified region is to locate the bins that overlap the region. Then each alignment in the bins is tested to check overlap. For example, in Table 6.1 a binning schema consisting of three types of bins was designed for a genome shorter than 288kbp. Bin 0 spans 0-288kbp, bin 1, 2 and 3 span 96kbp and bins from 4 to 12 span 32kbp each. An alignment beginning at 130kbp and ending at 134kbp [130k, 134k] was stored in bin 8. Similarly, an alignment [102k, 140k] was contained in bin 2, while an alignment between [80k, 98k] in bin 0. If all the alignments overlapping region [130k, 142k] are searched, bin 0, 2 and 8 will be located and fully scanned to find the required alignments.

These multi-dimensional range queries in databases are topics gaining attentions for more than 20 years. R-tree, R+-tree [SRF87], and their successors extend B-tree, divide the

0 (0-288kp)								
1 (0-96kp)			2 (96-192kp)				3 (192-288kp)	
4 (0-32kp)	5 (32-64kp)	6 (64-96kp)	7 (96-128kp)	8 (128-160kp)	9	10	11	12

Table 6.1: BAM Index

multi-dimensional space, and store the recursively divided spaces as tree nodes. Queries walk through the tree to find out the data block. These schemes does not consider the reliability problem over DOT, unless they are implemented in a scalable and reliable way, just like the distributed B-tree [MMN<sup>+</sup>04]. Even though, the performance degradation also exists due to missing clustered data.

### 6.3 Key Value Model

HBase is implemented on a LSM Tree, which only supports query based on Row Key instead of dual dimensional (2D) queries powered by R-tree. Inspired from the UCSC binning index, I designed a new binning schema to both support alignments retrieval overlapping a specific region and take full advantage of HBase fast scan operations.

One popular way to solve this problem is to reduce the dimensionality by searching in one-dimension firstly to find the minimum range containing all possible result and then using the other dimension to filter unexpected alignments. In this case, the starting base position minus the maximum mapped length (MML) could be a Row Key to find all possible alignments overlapping a specified range. The MML of a dataset is calculated from CIGAR of each read and stored in a table. For example, searching range [start, end] can be converted into query [start - MML, end] on the starting index. If starting position minus MML is below zero, zero will be the starting point. Then the ending position can filter the unexpected alignments. When searching a region in the other bin, searching range [start, end] can be converted into query [start - MML, end] on the starting index first. Then the end position filters unexpected alignments.

There are many methods [APH14, ARH12, GHR12a] and tools (baySeq [HK10], BBSeq [ZXW11], NOISeq [TGAD<sup>+</sup>11], QuasiSeq [LNM<sup>+</sup>12], etc.) to count reads. Some of the methods count them by pair and others by single read. Thus, all alignment in this Key Value schema are stored and searched by read in order to meet the requirements from both kinds of methods.

A alignment Key Value schema has been implemented according to generic data model in order to provide fast query over alignment data. Unlike a normal HBase table, which has at least three data copies in HDFS, each table in this design has only one copy. The three tables together complete the data consistence and fault tolerance. This design allows for high-speed queries for three different purposes without needing to use any extra storage space. To compensate, a manual data consistency check is required on changing data. However, data stored in transSMART is updated infrequently and one of its main aims is to provide fast data queries over the database.

Three tables are created in this schema to provide fast query over alignments overlapping a certain region to calculate gene or trait expression level for clinical decision making, as shown in Figure 6.2. The Subject table serves for queries over subject in a study, the Position table is used for for queries over reference sequence positions in a study and the Cross-study table supports queries over reference sequence positions in multiple studies.

In each table there are two parts in a dataset, header information and alignment data. The Row Key of the header in Subject table is a composite key containing a subject identity and a HTC. All header records use the Family header. The qualifier represents each TAG in the corresponding header line. All mandatory fields in the alignment are stored in Family align, while the optional fields uses Family opt. Not all the mandatory fields are required all the time. Some Qualifiers, which are more frequently used than others, can have their own families to accelerate specific queries. For example, MAPQ mapping quality is a most frequently used filter. Then MAPQ can be stored in a new

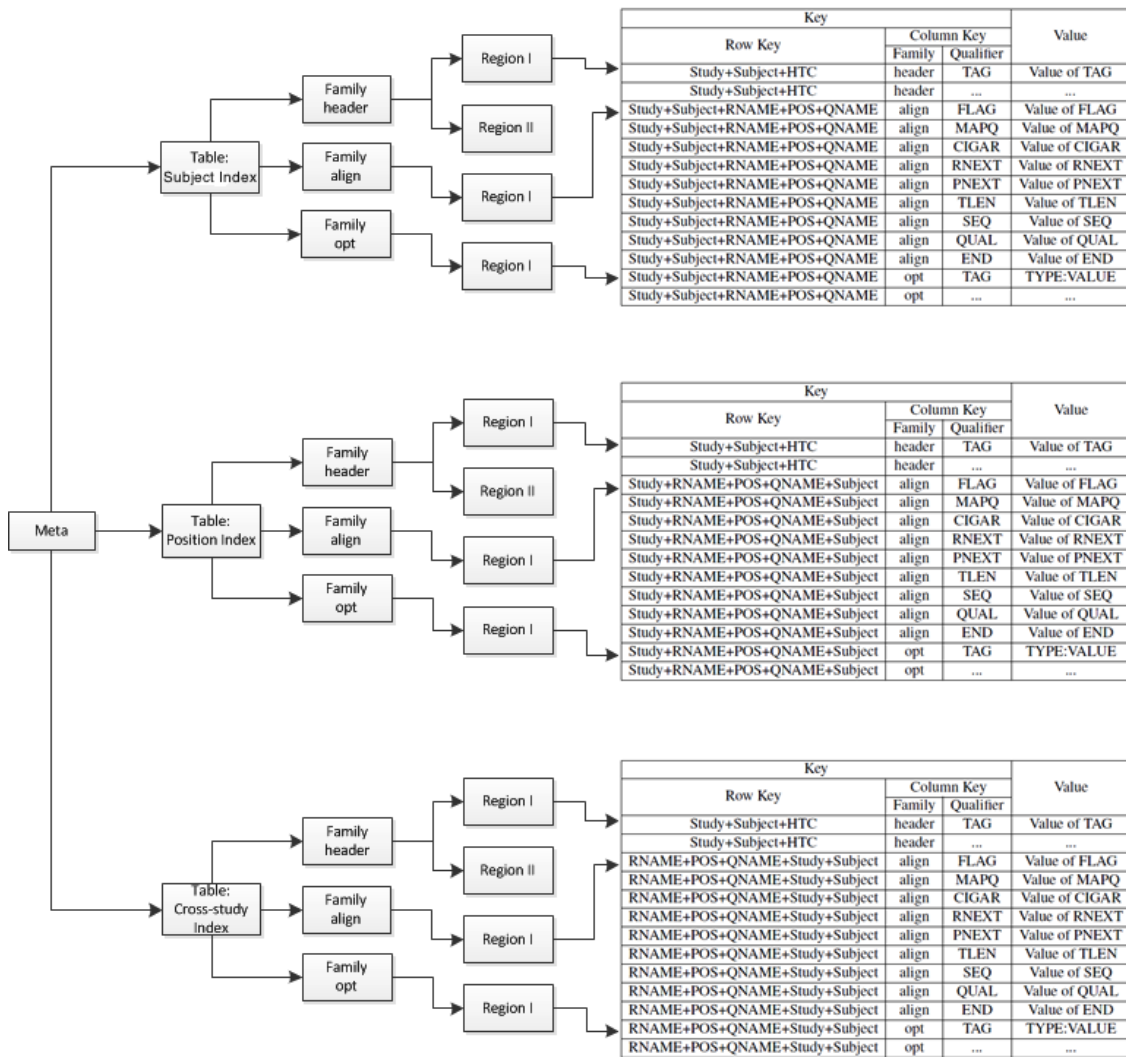


Figure 6.2: Raw sequencing Key Value model.

Family named mapq. In that case, MAPQ may have two copies in both Family align and mapq or only one copy in Family mapq. As described in Chapter 3, different Families are store in different HFiles. If only Family mapq is added to a Scan operation, only the corresponding HFiles are searched. Thus, the query with Family mapq will be much faster than the query with Family align. Furthermore, multiple such small Families may be used together to provide faster queries.



### 6.3.1 Subject table

As shown in Table 6.2, Study, Subject and HTC consist of the Row Key for Key Value pairs in Family header. The header TAG is used as Qualifier. Each Value contains the value of each TAG. Study, Subject, RNAME, POS and QNAME are comprised of the Row Key for alignment data in both Family align and Family opt. The Qualifiers with Family align are mandatory fields in SAM alignment structure, such as FLAG, RNAME, etc. An alignment row with optional information places each optional TAG as a Qualifier and the corresponding Value of each TAG use format TYPE:VALUE. In addition, a pair of paired-end reads share a same QNAME. Thus, each row in this QNAME table has two versions when loading paired-end reads.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Study+Subject+HTC	header	TAG	Value of TAG
Study+Subject+HTC	header	...	...
Study+Subject+RNAME+POS+QNAME	align	FLAG	Value of FLAG
Study+Subject+RNAME+POS+QNAME	align	MAPQ	Value of MAPQ
Study+Subject+RNAME+POS+QNAME	align	CIGAR	Value of CIGAR
Study+Subject+RNAME+POS+QNAME	align	RNEXT	Value of RNEXT
Study+Subject+RNAME+POS+QNAME	align	PNEXT	Value of PNEXT
Study+Subject+RNAME+POS+QNAME	align	TLEN	Value of TLEN
Study+Subject+RNAME+POS+QNAME	align	SEQ	Value of SEQ
Study+Subject+RNAME+POS+QNAME	align	QUAL	Value of QUAL
Study+Subject+RNAME+POS+QNAME	align	END	Value of END
Study+Subject+RNAME+POS+QNAME	opt	TAG	TYPE:VALUE
Study+Subject+RNAME+POS+QNAME	opt	...	...

Table 6.2: Alignment Key Value Subject table

#### Determining Row Key and Column Key of transcriptomic data

In order to speed up queries, the Study, Subject and HTC are comprised of the Row Key for data in Family header. HTC is the unique main field in the header that can be used as

Key. Study, Subject, RNAME, POS and QNAME are comprised of the Row Key for data in Family align and opt. All the other fields are placed in the Column Key. For a typical use case when multiple subjects within a study are retrieved, the HFile storing the data will be loaded into memory caches. Therefore, only retrieving the first record causes a significant delay when loading data from disk to memory, while all other records in the same HFile are fetched directly from memory caches. This design takes full advantage of the caches.

### **Optimizing the Column Key to increase cache hit rate**

One way to further increase cache hit rate is to design a new structure by classifying different types of data under Column Key. The Family divides different types of data (header, align and opt) into different HFiles and the Qualifier orders the general information and subject values lexicographically.

### **6.3.2 Position Table**

This Subject table only provides faster queries over Subject within one study. However, if queries by reference position in a study are required, these queries may not be performed as fast as the queries over Subject. Thus, the Position table is created to improve the performance for these queries.

As shown in Table 6.3, the header in the Position table uses the same schema as that in the Subject table in order to keep enough data copy for the fault tolerance. Study, RNAME, POS, QNAME and Subject are comprised of the Row Key for alignment data in both Family align and Family opt. The Qualifiers with Family align are mandatory fields in SAM alignment structure, such as FLAG, RNAME, etc. In addition, an END field (ending position in a reference) calculated from CIGAR is added to Family align.

An alignment row with optional information places each optional TAG as a Qualifier and the corresponding Value of each TAG use format TYPE:VALUE.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Study+Subject+HTC	header	TAG	Value of TAG
Study+Subject+HTC	header	...	...
Study+RNAME+POS+QNAME+Subject	align	FLAG	Value of FLAG
Study+RNAME+POS+QNAME+Subject	align	MAPQ	Value of MAPQ
Study+RNAME+POS+QNAME+Subject	align	CIGAR	Value of CIGAR
Study+RNAME+POS+QNAME+Subject	align	RNEXT	Value of RNEXT
Study+RNAME+POS+QNAME+Subject	align	PNEXT	Value of PNEXT
Study+RNAME+POS+QNAME+Subject	align	TLEN	Value of TLEN
Study+RNAME+POS+QNAME+Subject	align	SEQ	Value of SEQ
Study+RNAME+POS+QNAME+Subject	align	QUAL	Value of QUAL
Study+RNAME+POS+QNAME+Subject	align	END	Value of END
Study+RNAME+POS+QNAME+Subject	opt	TAG	TYPE:VALUE
Study+RNAME+POS+QNAME+Subject	opt	...	...

Table 6.3: Alignment Key Value Position table

### Determining Row Key and Column Key of transcriptomic data

To speed up queries over gene positions, the Study, Subject, RNAME and POS should be placed in the Row Key. The reason is that RNAME with POS is identical information, as mentioned in the Subject table section. But RNAME with POS is not unique. Row Key should be a unique key. Thus, QNAME has been added at the end of Row Key to make Row Key unique. A second reason is that when a region is retrieved in a typical use case, the HFiles storing the data in these positions will be loaded into MemStore. Thus, only retrieving the first record leads to a long delay due to fetching data from disk to memory, while all other records in the same HFile are loaded directly from memory caches. This design speeds up queries by taking advantages of the caches.

### **Optimizing the Row Key to speed up data location**

The order of the fields in the Row Key also greatly affects the query performance. Typical user cases may start with a gene information query in a study. By placing the Subject after the reference attributes (QNAME, RNAME and POS), the composite key becomes two indices on the subject and genetic information respectively. By placing RNAME with POS before QNAME, the composite key becomes three indices on the subject, reference sequence position and sequence template name respectively.

When alignments overlapping a specified region [start, end] in a subject are retrieved, this Row Key can find all the possible HFiles using this Row Key to compare to the (start - MML) and End. Then the END value of this alignment determines whether this alignment is required.

### **6.3.3 Cross-study Table**

The Position table only provides faster queries over reference positions within one study. However, if queries across studies are required, these queries may not be performed as fast as the queries within one study. Therefore, a cross-study table has been created to overcome this disadvantage.

As shown in Table 6.4, the Cross-subject table is very similar to the Position table except the order of the fields in the Row Key. The header in this table is also used to keep enough data copy for the fault tolerance of the header data.

### **Optimizing the Row Key to speed up data location**

The order of the fields in the Row Key focuses on the performance of queries across studies. Typical user cases may start with a gene information query across studies. By

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Study+Subject+HTC	header	TAG	Value of TAG
Study+Subject+HTC	header	...	...
RNAME+POS+QNAME+Study+Subject	align	FLAG	Value of FLAG
RNAME+POS+QNAME+Study+Subject	align	MAPQ	Value of MAPQ
RNAME+POS+QNAME+Study+Subject	align	CIGAR	Value of CIGAR
RNAME+POS+QNAME+Study+Subject	align	RNEXT	Value of RNEXT
RNAME+POS+QNAME+Study+Subject	align	PNEXT	Value of PNEXT
RNAME+POS+QNAME+Study+Subject	align	TLEN	Value of TLEN
RNAME+POS+QNAME+Study+Subject	align	SEQ	Value of SEQ
RNAME+POS+QNAME+Study+Subject	align	QUAL	Value of QUAL
RNAME+POS+QNAME+Study+Subject	align	END	Value of END
RNAME+POS+QNAME+Study+Subject	opt	TAG	TYPE:VALUE
RNAME+POS+QNAME+Study+Subject	opt	...	...

Table 6.4: Alignment Key Value Cross-study table

placing the Study and Subject after the reference position (RNAME, POS and QNAME), the composite key becomes an efficient index for cross-subject gene location.

When alignments overlapping a specified region [start, end] within several are retrieved, this Row Key can find all the possible HFiles using this Row Key to compare to the (POS - MML) and END.

## 6.4 Performance Evaluation

### 6.4.1 Test environment and datasets

A datasets HG00097CH11 including about 13 million reads from 1000 Genome project [Siv08] was used to test this new Key Value data schema on a cloud computing platform, OpenStack, installed at Imperial College London. In this experiment Samtools was installed in a VM with 32 CPU cores and 32 GB of memory for queries over BAM files. A simple socket based server implemented in Java was used as BAM server to respond

alignment queries. The HBase cluster (Version 0.96.0 on Hadoop 1.0.3) is comprised of 4 virtual machines, each with 8 CPU cores and 8 GB of memory. Each VM used a 100 GB disk. HBase was used to implement the Key Value model. An extra Family count, including MAPQ and END, is added to accelerate queries for gene expression calculation. The other fields in the original Family align are still stored in align. The BAM-based model is also implemented in HBase. The STUDY, SUBJECT and a unique column ID are added to the basic 11 SAM columns. The ID is the Row Key. The STUDY, SUBJECT and RNAME+POS are indexed using the secondary index. The value in the index tables is the ID. One master server and four region servers in HBase are configured in fully distributed mode. The master server and one region server reside in one VM. The other region servers reside in the other three VMs respectively. In addition, one independent VM with 8 CPU cores and 8 GB or memory was used as a test client to send query requests to and receive data from both the BAM server and the HBase cluster.

#### **6.4.2 Query over alignments overlapping specified regions**

Query over alignments overlapping specified regions is the most important operation for a RNA-Seq based expression level calculation because many regions are linked with influencing disease risk. For example, about 18 regions of the genome have been linked with influencing type 1 diabetes risk [SBAJ<sup>+</sup>09]. These regions, each of which may contain several genes, have been labeled from IDDM1 to IDDM18. IDDM1 contains the HLA genes and IDDM2 contains the insulin gene INS. They were both originally identified by investigating the suspected genes, HLA genes and INS, respectively, using case control studies. The remaining type 1 diabetes susceptibility loci, IDDM3 to IDDM18, were mainly discovered by genome scan linkage studies, for example, looking for linkage between regions of the genome and disease in affected sib-pairs. Many genes are mapped to the loci, such as IL2RA and CTLA4. Except loci IDDM1 to IDDM18, other genes as-

sociates type 1 diabetes as well. For example, gene IFIH1 associates with different rates of progression from autoimmunity to type 1 diabetes.

In order to improve risk prediction of a specific disease, most corresponding linked regions are collected for the calculation. These genes are distributed in multiple places with various lengths. For example, in type 1 diabetes, INS spans about 4000 bases in chromosome 11, HLA-DQB1 spans around 7000 bases in chromosome 6, IL2RA spans around 50,000 bases in chromosome 10, CTLA4 spans about 6000 bases in chromosome 2 and IFIH1 spans about 52,000 bases in chromosome 2. The longest human gene DMD spans more than 2,000,000 bases.

Therefore, test cases using 5,000, 10,000, 50,000, 100,000 and 500,000 bases are generated to fully test this Key Value schema. In each case the region is searched concurrently using from 5, 10, 50 and 100 threads. The locations of these regions are generated randomly.

With the thread number less than 10 and the region size less than 500,000, the Key Value model in HBase outperforms the SAM model in HBase by 72.63 times on average. With thread number and region size increasing, the advantage of the Key Value model is more significant. Especially, in 100,000-base cases the Key Value implementation performs more than 115 times faster than the SAM model in HBase. Though both the Key Value model in HBase and BAM server responded every request within one second, the Key Value model in HBase outperformed the BAM server in the 10 tests out of the total 12 tests, as shown in Figure 6.3. Especially in the test with 10 threads and 100,000 bases, the Key Value model in HBase performed more than 2 times faster than the BAM server.

With concurrent query number increasing, the BAM server did not scale well. While more than 50 concurrent requests were arrived with larger regions, the Key Value model greatly outperformed the BAM server, as shown in Figure 6.4. In 7/10 test cases, the Key Value model performed more than 3 times faster than the BAM server. In 4/10 cases,

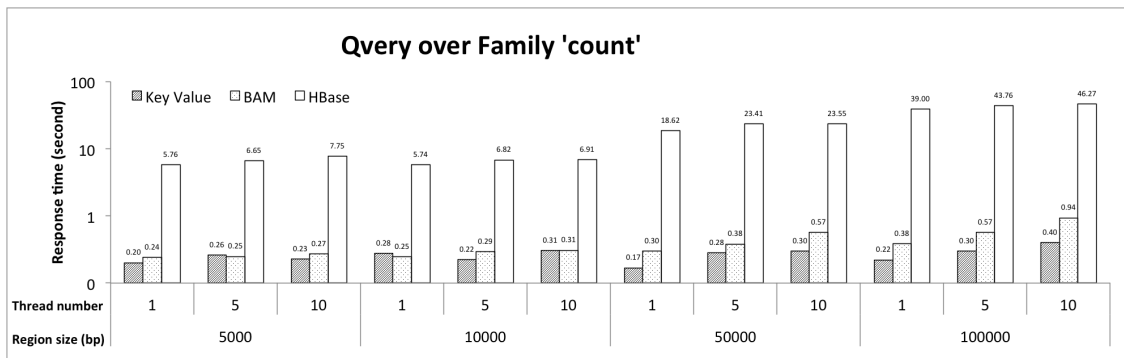


Figure 6.3: Alignment query using less than 10 threads. The region size is less than 100,000 bp.

HBase outperformed the BAM server by more than 5 times. In the 500,000 base tests, the Key Value model performed more than 9 times faster than the BAM server. In the largest case the BAM server execution time increased largely, up to 37 seconds. Compared to the BAM model in HBase, the Key Value model shows an average 76.78-fold speedup. Especially, in the 500,000 base tests, the Key Value model outperforms the BAM model in HBase by more than 440 times.

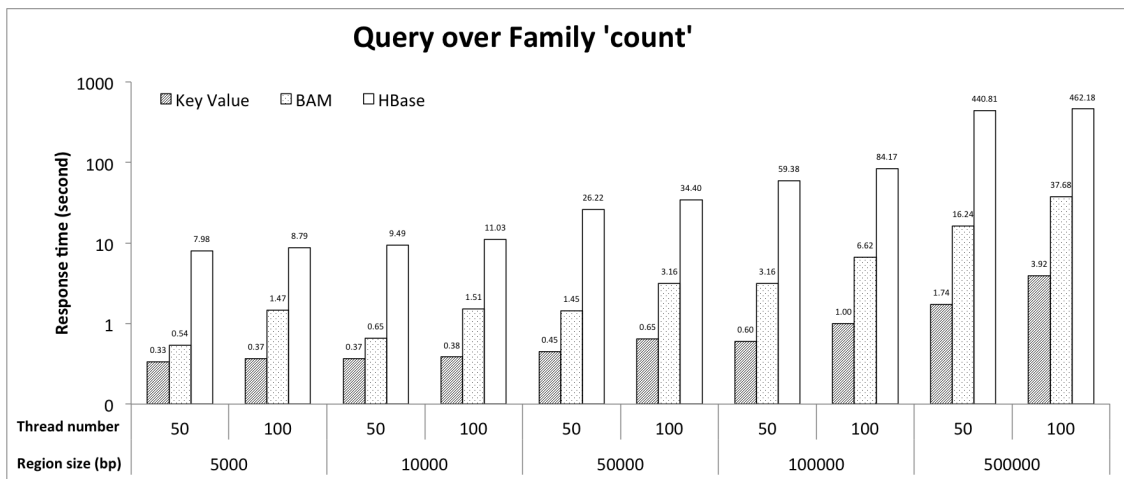


Figure 6.4: Alignment query using more than 50 threads. The maximum region size is 500,000 bp.

In sum, the response time of the Key Value model was no longer than the BAM server in most cases, except 5-thread-5,000-base and 1-thread-10,000-base. The reason is that some of the small tests in 5-thread-5,000-base and 1-thread-10,000-base cases suffer from long time responses of HBase. The probable reason is the periodical HBase system main-



tenance that could lead to hundreds of milliseconds delay. Also, the execute time did not rise greatly when thread number increased and the longest execution time is only less than 4 seconds, 9.61 times faster than BAM server, as shown in Figure 6.4. This mainly results from the special Family count design. As shown in Figure 6.5, when concurrent queries were executed, the queries with Family align performed even worse than BAM. But when a specific Family used, such as count, HBase outperformed BAM by up to 6 times.

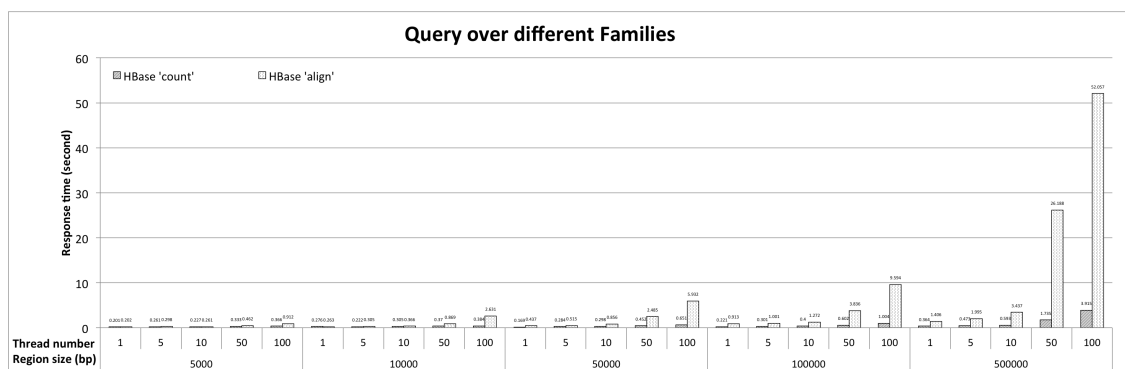


Figure 6.5: Alignment query with different Families. The maximum thread number is 100 and the maximum region size is 500,000 bp.

## 6.5 Integration into tranSMART

This implementation is based on HBase 0.96.0. As shown in the Figure 2.11, the patient information is retrieved from a SQL database before a variant data query. Three types of data queries are created corresponding to the three complementary tables. Figure 6.6 shows the alignment data query overlapping a specific region within a subject using the Scan method; Figure 6.7 indicates the alignment data query overlapping a specific region across subjects using the sequential Scan method; Figure 6.8 describes the alignment data query method by RNAME using the Scan method. The cacheSize used in the last two Figures may be changed according to the physical environments and the data scale.

```

/*
 * Scan alignment records overlapping a specific region within a subject
 * @parameter subject: the name of the subject
 * @parameter rname: the value of RNAME
 * @parameter start: the genetic starting position
 * @parameter end: the genetic ending position
 * @parameter cache: the buffer size of HBase scanner
 */
overlappingScan(subject, rname, start, end, cache) {
    List results;
    Scan s = new Scan();
    s.setCacheBlocks(true);
    s.setCaching(cache);
    startRow = subject + rname + start;
    endRow = subject + rname + end;
    s.setStartRow(startRow);
    s.setStopRow(endRow);
    ResultScanner scanner;
    scanner = alignPosTable.getScanner(s);
    for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
        for (Cell kvPair : rr.rawCells()) {
            results.add(kvPair);
        }
    }
}

```

Figure 6.6: Scan overlapping a specific region within a subject.

```

/*
 * Scan alignment records overlapping a specific region across subjects
 * @parameter subject: the name of the subject
 * @parameter rname: the value of RNAME
 * @parameter start: the genetic starting position
 * @parameter end: the genetic ending position
 * @parameter cache: the buffer size of HBase scanner
 */
overlappingScan(subject, rname, start, end, cache) {
    List results;
    Scan s = new Scan();
    s.setCacheBlocks(true);
    s.setCaching(cache);
    startRow = start + subject + rname;
    endRow = end + subject + rname;
    s.setStartRow(startRow);
    s.setStopRow(endRow);
    ResultScanner scanner;
    scanner = alignAcrossTable.getScanner(s);
    for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
        for (Cell kvPair : rr.rawCells()) {
            results.add(kvPair);
        }
    }
}

```

Figure 6.7: Scan alignment records overlapping a specific region across subjects.

## 6.6 Discussion

### 6.6.1 Family organization

If a Family becomes smaller, query over the Family will be much faster than a larger one. But when many Families are searched concurrently, the combination of smaller

```
/*
 * Random read alignment records by RNAME
 * @parameter subject: the name of the subject
 * @parameter rname: the value of RNAME
 */
getRecordByRname (subject, rname) {
    List results;
    Get get = new Get(subject + rname);
    Result result = shortQnameTable.get(get);
    for (Cell kvPair : result.rawCells()) {
        results.add(kvPair);
    }
}
```

Figure 6.8: Random read alignment records by RNAME.

Families may be even slower than the larger one. Thus, the Family organization should be determined by applications. For example, in the gene expression application, except fields stored in the Row Key, MAPQ and END may be the most useful information and stored in a Family named count.

### 6.6.2 Extra disk space

The Key Value model is a space for time approach and gains a lot of velocity. The current space overhead is large. For example, in the performance evaluation section the BAM file size is 1.29 GB (the corresponding SAM is 5.69 GB), but HBase used 29.16 GB to store three copies of the data. Each used 7.53 times of storage space of BAM. The main reason for this large overhead is the column number. If a row contains N columns, the Row Key will be duplicated N-1 times. Further the composite Row Key in the Key Value model contains multiple columns, which lead to a even larger overhead. A possible way to reduce this overhead is to merge columns. If some columns are frequently used together, such RNEXT and PNEXT, it is possible to reduce one Row Key space by merging column RNEXT and column PNEXT into column RNEXT+PNEXT.

### 6.6.3 Long MML binning schema

In the most popular NGS sequencing machines, such as Illumina [Gen10] and SOLiD [Dek07], the length of a RNA-Seq read is usually less than 300 bp.

For example, the alignment dataset HG00096CH11 published in 1000 Genomes only contains reads mapped to less than 200 bases. But if long mapped reads are emerging, the scan method used in this Chapter may not be suitable. The whole dataset can be divided into two bins: one containing reads whose MML is shorter than a certain number CLEN and the other one containing all remaining alignments, including alignments with MML not shorter than CLEN bases. Then a Long MML schema is required.

There are tables in this schema for fast query over alignments overlapping a certain region. Two Position tables (Starting position and Ending position) serve for queries over reference sequence positions in a subject and two Cross-subject tables (Starting cross-subject and Ending cross-subject) support queries over reference sequence positions in multiple subjects. The Starting position table uses the same schema as Position table in Table 6.3. The Starting cross-subject table uses the same schema as Cross-subject table in Table 6.4. The ending position is the rightmost mapped base for most of the read. However, if a read is mapped to different reference sequences, the ending position is the ending position of the next reference sequence. If the next reference information is unknown, the ending position is the ending position of the current segment.

#### A pair of Position tables

The Starting position table uses the same schema as Position table in Table 6.3. The Ending position table uses similar schema to the Starting position table, as shown in Table 6.5.

The header in the reference index table uses the same schema as that in the Position table

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Subject+HTC	header	TAG	Value of TAG
Subject+HTC	header	...	...
Subject+RNAME+END+QNAME	align	FLAG	Value of FLAG
Subject+RNAME+END+QNAME	align	MAPQ	Value of MAPQ
Subject+RNAME+END+QNAME	align	CIGAR	Value of CIGAR
Subject+RNAME+END+QNAME	align	RNEXT	Value of RNEXT
Subject+RNAME+END+QNAME	align	PNEXT	Value of PNEXT
Subject+RNAME+END+QNAME	align	TLEN	Value of TLEN
Subject+RNAME+END+QNAME	align	SEQ	Value of SEQ
Subject+RNAME+END+QNAME	align	QUAL	Value of QUAL
Subject+RNAME+END+QNAME	align	END	Value of QUAL
Subject+RNAME+END+QNAME	opt	TAG	TYPE:VALUE
Subject+RNAME+END+QNAME	opt	...	...

Table 6.5: Alignment Key Value Position END table

in order to keep enough data copy for the fault tolerance. In the alignment part DATASET, RNAME, END and QNAME are composed of the key. The value includes Family align and opt, which are the same as Table 6.3.

Associated with Starting position table, the pair position tables can speed up queries over gene positions. When searching a region [start, end] in this bin, there are two methods to find all possible alignments: searching [start - MML, end] in the Starting position table and searching [start, end + MML] in the Ending position table. If the MML is very close to the length of the reference sequence, query over [start - MML, end] using the Starting position table usually means searching [0, end]. The situation of Ending position table is the same as Starting position table. Thus, if the middle position of the region resides ahead of the middle position of the bin, most of the expected alignments should reside in the first half of bin. In that case, Starting position table should be searched. Otherwise, it is better to search Ending position table.

### A pair of Cross-subject table

The pair of Position tables can only offer faster queries by reference position within one subject. If queries over multiple subjects are required, these queries may not be performed as fast as the queries within one subject. Therefore, a pair of Cross-subject tables has been created to overcome this disadvantage.

The Starting cross-subject table uses the same schema as Table 6.4. The Ending cross-subject table uses similar schema to the Starting cross-subject table, as shown in Table 6.6.

Key			Value
Row Key	Column Key		
	Family	Qualifier	
Subject+HTC	header	TAG	Value of TAG
Subject+HTC	header	...	...
RNAME+END+QNAME+Subject	align	FLAG	Value of FLAG
RNAME+END+QNAME+Subject	align	MAPQ	Value of MAPQ
RNAME+END+QNAME+Subject	align	CIGAR	Value of CIGAR
RNAME+END+QNAME+Subject	align	RNEXT	Value of RNEXT
RNAME+END+QNAME+Subject	align	PNEXT	Value of PNEXT
RNAME+END+QNAME+Subject	align	TLEN	Value of TLEN
RNAME+END+QNAME+Subject	align	SEQ	Value of SEQ
RNAME+END+QNAME+Subject	align	QUAL	Value of QUAL
RNAME+END+QNAME+Subject	align	END	Value of QUAL
RNAME+END+QNAME+Subject	opt	TAG	TYPE:VALUE
RNAME+END+QNAME+Subject	opt	...	...

Table 6.6: Alignment Key Value Cross-subject END table

The Cross-subject END table is very similar to the position table except the order of the fields in the Row Key. The header in this table is still used to keep enough data copy for the fault tolerance of the header data.

Associated with Starting cross-study table, the pair of cross-study tables provides fast queries over gene positions across multiple subjects. The same searching strategy as the pair of position tables can be applied on this pair of cross-subject tables.

When searching a region in the long TLEN bin, the average position  $((\text{start} + \text{end}) / 2)$  determines starting or ending index should be used. The middle position, which cuts the reads into 2 bins each containing the same amount of alignments, is recorded after datasets imported. If the MML is very close to the length of the reference sequence, query over  $[\text{start} - \text{MML}, \text{end}]$  using the starting position index usually means searching  $[0, \text{end}]$ . The situation of ending position tables is very similar. Thus, if the average position is located in the first half of alignments, the starting tables will be utilized. Otherwise, the ending tables will be used. The average scan range is  $1/4$  positions and the worst case is  $1/2$  positions.

## 6.7 Conclusion

This chapter describes a Key Value data model for queries over raw sequence data, compared to the SAM data format implemented in HBase and a BAM server. In the performance evaluation experiments, the Key Value data model performs up to 440 and 22 times faster than the SAM format implemented in HBase and the BAM server.

# Chapter 7

## Gene expression based patient stratification application

### 7.1 Introduction

Information from genomic, proteomic and metabolic measurements benefits molecular profiling based patient stratification. Biomedical research is moving towards using high-throughput molecular profiling data to improve clinical decision making. One approach for building classifiers is to stratify subjects based on their molecular profiles. Unsupervised clustering algorithms can be used for stratification purposes. This chapter introduces a hierarchical clustering application using four kinds of correlation methods with high-dimensional molecular profiling data (gene expression data), by taking full advantage of a programming model specifically designed for parallel processing of big datasets.

The motivation for optimization of hierarchical clustering is based on the experiences in using tranSMART Advanced Workflow functions. For example, in Figure 7.1, two cohorts in the study GSE31773 [TWM<sup>+</sup>12] are selected, one for severe patients with CD4<sup>+</sup> T-cells and the other for severe patients with CD8<sup>+</sup> T-cells. [TWM<sup>+</sup>12] indicates



comparison of mRNA expression showed widespread changes in the circulating CD8+ but not CD4+ T-cells from patients with severe asthma. No changes were observed in the CD4+ and CD8+ T-cells in non-severe asthmatics versus healthy controls. Thus, severe patients with CD4+ T-cells and severe patients with CD8+ T-cells should be significantly different in their mRNA expression. The Hierarchical Clustering function of Advanced Workflow, as shown in Figure 7.2, can be used to validate this result. If [TWM<sup>+</sup>12] is correct, the two cohorts should be exactly divided into two groups. In this example, 50 probes are used to validate the result. The Figure 7.3 validated the results in [TWM<sup>+</sup>12]. To the opposite, healthy individuals and non-severe patients with CD4+ T-cells and those with CD8+ T-cells should not be significantly different in their mRNA expression, as shown in Figure 7.4. If using hierarchical clustering, the two subsets should not be divided into accurate two clusters in most probes. The Figure 7.5 validated the hypothesis.

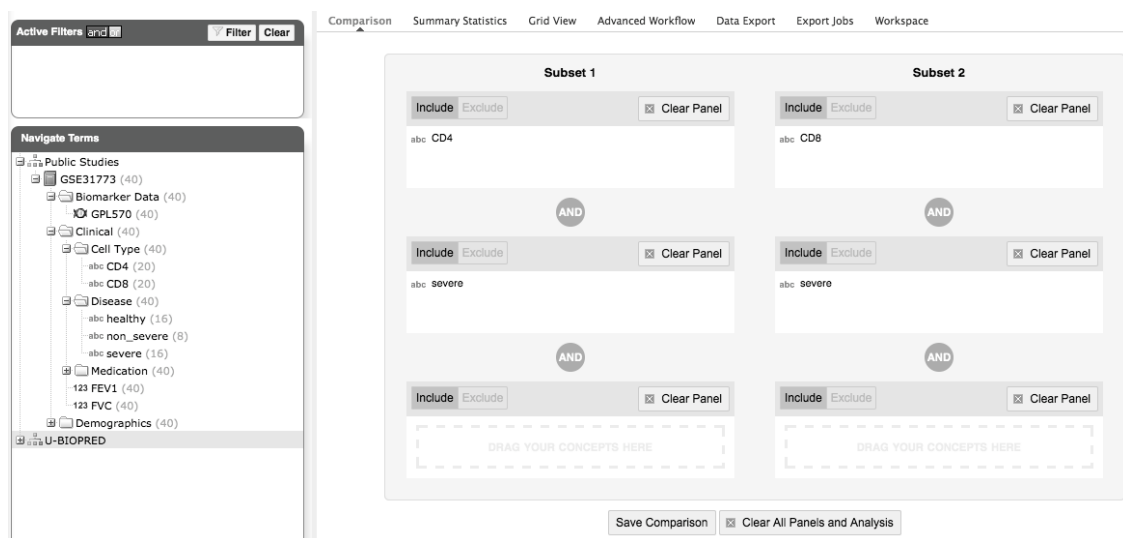


Figure 7.1: Cohorts selection of severe samples. In subset1 there are two condition boxes. The first box contains all CD4+ T-cells and the second box contains all severe samples. Thus, the subset1 contains all the samples who are both CD4+ T-cells and severe. Then, the subset2 contains all samples who are both CD8+ T-cells and severe.

The aim is to optimize transSMART so that clinicians can make use of it for faster clinical decision making. However, the current transSMART's performance of preparing correlation matrices when analyzing high dimensional molecular data is sub-standard.

Analysis: Hierarchical Clustering

Cohorts:  
 Subset 1: INCLUDE ( \Public Studies\GSE31773\Clinical\Cell Type\CD4\ ) AND INCLUDE ( \Public Studies\GSE31773\Clinical\Disease\severe\ )  
 Subset 2: INCLUDE ( \Public Studies\GSE31773\Clinical\Cell Type\CD8\ ) AND INCLUDE ( \Public Studies\GSE31773\Clinical\Disease\severe\ )

**Variable Selection** ?

Select a High Dimensional Data node from the Data Set Explorer Tree and drag it into the box.

GPL570

High Dimensional Data Clear

**GPL Platform:** GPL570  
**Sample:** CD8\_RNA  
**Tissue:**  
**Pathway:**  
**Probe aggregation:** false  
**Marker Type:** Gene Expression

Max rows to display: 50

Apply clustering for rows  
 Apply clustering for columns  
 Calculate z-score on the fly

Run

Figure 7.2: Hierarchical Clustering parameters. The cohorts are set in Figure 7.1. No gene is specified. So all probe data are loaded, but only the first 50 probes are shown in the heatmap. Clustering is applied for both rows and columns.

For example, an unsupervised hierarchical clustering has been performed on the publicly available MULTMYEL dataset taken from NCBI's GEO (GSE24080). The dataset contains 559 subjects' gene expression data produced by an Affymetrix GeneChip Human Genome U133 Plus 2.0 Array. In order to build the classifiers, the subjects are clustered using a hierarchical clustering algorithm implemented in R. In previous tests, running this algorithm on a virtual machine configured with 32 cores and 32 GB of member took over 6 minutes using a Euclidean distance matrix, and more than a week if performing a Kendall rank correlation. Compared to the correlation matrix calculation, the hierarchical clustering function in R only needs less than one second to execute. These observations show that the bottleneck in the hierarchical clustering algorithm is in generating the cor-

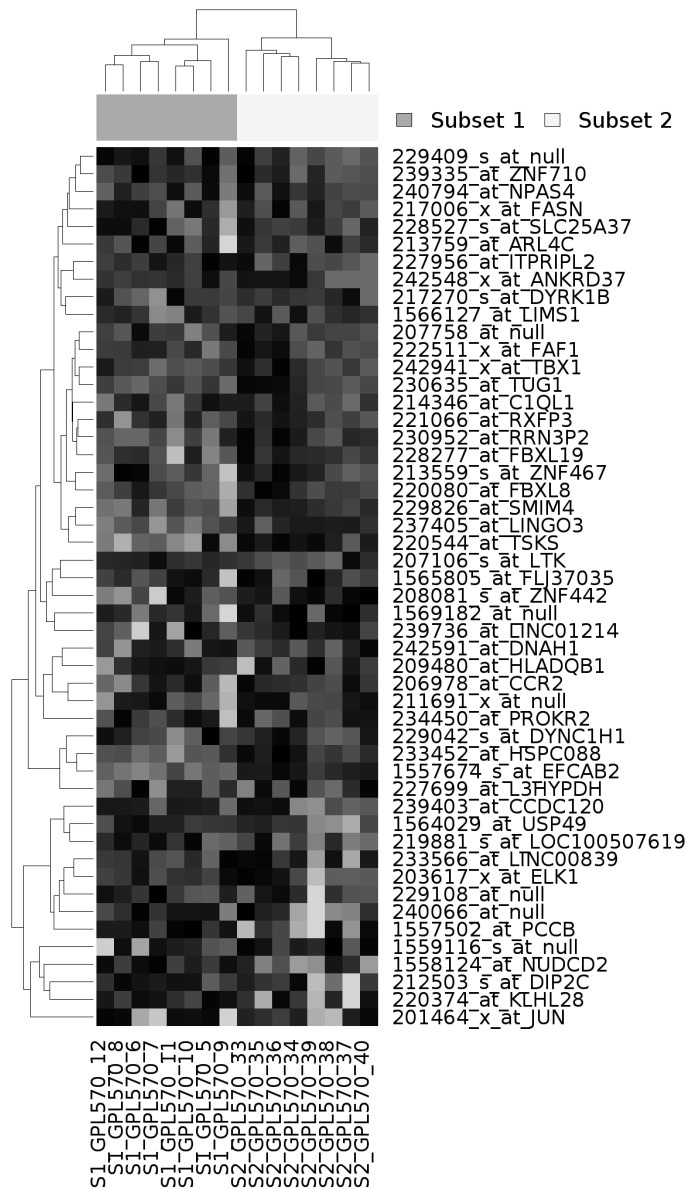


Figure 7.3: Hierarchical Clustering result of severe samples. The first 8 columns are samples in subset1 and the remaining 8 columns are samples in subset2. According to the clustering by column shown at the top of the graph, two subset are exactly divided.

relation matrix. Clearly optimizing the performance of these analyzes would expedite clinical decision making. With high throughput sequencing technologies promising to produce even larger datasets per subject, the performance of the state-of-the-art statistical algorithms is to be further impacted unless efforts towards optimization are carried out.

The image shows a web-based interface for selecting cohorts. It is divided into two main sections: 'Subset 1' and 'Subset 2'. Each section contains three vertically stacked panels. Each panel has a header with 'Include' and 'Exclude' buttons, and a 'Clear Panel' button with a close icon. Below the header is a text input area. In Subset 1, the first panel contains 'abc CD4', the second panel contains 'abc healthy' and 'abc non\_severe', and the third panel is a dashed box with the text 'DRAG YOUR CONCEPTS HERE'. In Subset 2, the first panel contains 'abc CD8', the second panel contains 'abc healthy' and 'abc non\_severe', and the third panel is a dashed box with the text 'DRAG YOUR CONCEPTS HERE'. Between the panels in each subset is a circular button with the text 'AND'. At the bottom of the interface are two buttons: 'Save Comparison' and 'Clear All Panels and Analysis' with a close icon.

Figure 7.4: Cohorts selection of not severe samples. In subset1 there are two condition boxes. The first box contains all CD4+ T-cells and the second box contains all healthy and non-severe samples. Thus, the subset1 contains all the samples who are healthy and non-severe CD4+ T-cells. Then, the subset2 contains all samples who are healthy and non-severe CD8+ T-cells.

In this chapter, an optimization method [WPJ<sup>+</sup>14] is provided for correlation calculations used by the hierarchical clustering algorithm in tranSMART. The optimization method contains a series of processing optimizations based around the MapReduce programming model [DG08]. The chapter also presents how the correlation matrix calculations implemented on R Rhipe [GHR<sup>+</sup>12b] plugin which works on Hadoop, a popular and well-supported distributed data storage and computation framework that supports MapReduce, significantly outperform their comparable implementations configured for distributed execution in R using Snowfall [KPB09], a parallel computing package for R scripts. The experiments use four correlation calculation methods, including the Pearson product-moment correlation [Wil96], Spearmans rank-order correlation [Spe04], Kendalls rank correlation [Ken48], and Euclidean distance correlation. In addition, the data in current tranSMART are stored in a Oracle or PostgreSQL database and moving to a big data

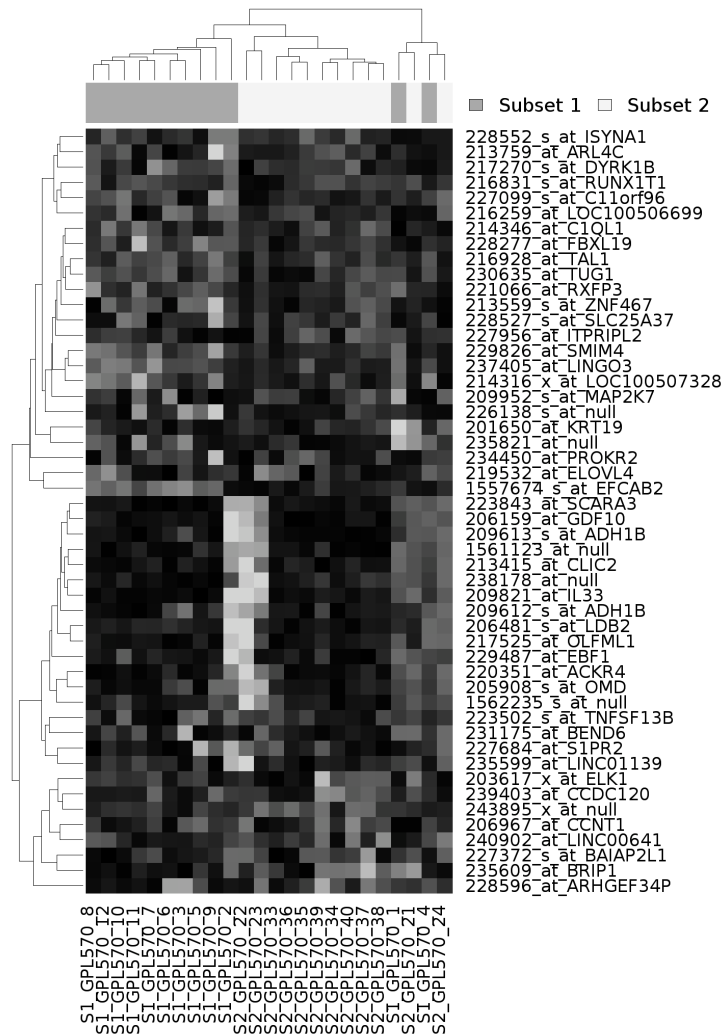


Figure 7.5: Hierarchical Clustering result of severe samples. The first 10 columns and 21st and 23rd columns are samples in subset1 and the remaining 12 columns are samples in subset2. According to the clustering by column shown at the top of the graph, two subset are not divided into two clusters.

warehouse, a mixed data storage, including traditional relational databases and new Key Value databases. Another experiment is carried out to show how the Key Value model affects the execution time of the whole analysis workflow.

## 7.2 Background

### 7.2.1 Correlation Algorithms

In data mining, hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters using correlation matrices. Currently there are three main types of correlation coefficient calculation algorithms: Product-moment correlation, rank correlation and other dependence measurements. Pearsons product-moment correlation is a measure of the linear correlation between two variables  $X$  and  $Y$ , giving a value between  $+1$  and  $-1$  inclusive, where  $1$  is total positive correlation,  $0$  is no correlation, and  $-1$  is total negative correlation. Spearman and Kendall correlation methods are two examples of using a rank correlation coefficient. Spearman's rank correlation coefficient is a non-parametric measure of statistical dependence between two variables by using a monotonic function. If there are no repeated data values, a perfect Spearman correlation of  $+1$  or  $-1$  occurs when each of the variables is a perfect monotone function of the other. Kendall tau rank correlation coefficient is a statistic used to measure the association between two measured quantities using a tau test, a non-parametric hypothesis test for statistical dependence based on the tau coefficient. Euclidean distance is a popular dependence measurement that differs from correlations by calculating the distance between two points according Pythagoras theorem within a metric space.

### 7.2.2 Parallel R packages

R is a widely used analysis tool for clustering and correlation calculation. Many emerging parallel R packages, such as RHIPE, SparkR [ZCF<sup>+</sup>10], RABID [LYM], Snowfall, Rmpi and pbdMPI [COS<sup>+</sup>12], can be used to parallelize R processes. Snowfall is a conventional parallel plugin that works in a master-slave mode, where all code and data are distributed to each process within a cluster, then the code works respectively in each process, and

finally the master collects the final result. Rhipe is a MapReduce based R plugin that transforms R functions into MapReduce jobs.

R is a widely used analysis tool for clustering and correlation calculation. RHIPE is a Hadoop MapReduce based R package that transforms R functions into MapReduce jobs. SparkR and RABID are MapReduce packages, which works in combination with Apache Spark [20]. Though Hadoop performs slower than Spark in several cases, such as iterative computations, Hadoop is much more mature to provide a more stable performance. Snowfall with Rmpi is a combination of parallel packages that works in a masterslave interactive mode, where all code and data are distributed to each process within a cluster, then the code works respectively in each process, and finally the master collects the final result. While pbdMPI uses Single Program Multiple Data parallel programming model, which is not as popular as Snowfall.

MapReduce is a simple processing model based around two functions that execute at two distinct stages: the mapper function, `Map()`, transforms raw input data into a set of intermediate Key Value pairs; then the reducer function, `Reduce()`, takes the related data (usually sorted by the key in the Key Value pairs emitted by the mapper) and merges the data together. The MapReduce task division is based around how the input data is distributed and processed by the mapper and reducer at each stage. Initially the input data is split and distributed to mappers that run on different compute nodes. After emitting some Key Value pairs, the MapReduce system sorts the pairs into related groups by key. These groups are then each provided as input to reducers that also run on different compute nodes. Finally, the output of the reducers is collected and combined into a single output dataset.

Snowfall and RHadoop are two R packages widely used to parallelize single-process R functions to achieve higher calculation speed. Snowfall supports socket connections and the Message Passing Interface (MPI). Snowfall operates MPICH cluster, an MPI imple-

mentation, through LAM/MPI [SL03]. RHadoop supports several Apache Hadoop sub-projects, including HDFS, MapReduce and HBase.

Rmpi [Yu02] allows users to create R programs that run cooperatively in parallel across multiple machines, or multiple CPUs on one machine, to accomplish a goal more quickly than running a single program on one machine.

## 7.3 Parallel R approaches

There are two classical types of R parallelizing methods, parallelizing independent for loop functions in R statements or splitting input data matrices.

### 7.3.1 Statement level parallelizing

The statement level parallelizing method can only be applied on independent for loop statements written in R. Thus, this method is widely used and very simple to implement.

For example, POPGEN [KNV<sup>+</sup>06] is a R package used for genotype clustering that focuses on statistical and population genetics methods. The motivation behind the package is to produce an easy to use interface to many of the commonly used methods and models used in statistical and population genetics and an alternative interface for some of the methodology produced by our group. It suffers from CPU and memory limitations due to being designed as a single computing process, and consumes a large amount of memory. For example, a variant value matrix for 100,000 SNPs with 10,000 subjects consumes 10 GB memory. If the number of SNPs or subjects increases, the memory requirement rises sharply to an amount that a single machine cannot support.



## Design

In the statement level a loop function including the clustering methods is re-written to a new function to be parallelized by Snowfall `sfLapply` functions. For example, in POPGEN `nps` function there is a loop statement that contains 100 independent clustering functions by default, as shown in Figure 7.6. This for loop function is replaced with a `sfLapply` function. In this `sfLapply` the `fun` parameter is filled by a new function including all codes in the for loop bracket. Thus, the `sfLapply` is able to parallelize the for loop by executing many instances of the new function concurrently.

## Performance evaluation

As shown in the Figure 7.7, each dataset contains different number of subjects, from 75 to 600, but all the other parameters are the same, including the number of unlinked loci (30,000), the number of alleles (2) at each locus and genotype data distribution. 4 VM, with each 8 virtual CPU cores and 8 GB memory, are used to deploy the LAM/MPI cluster.

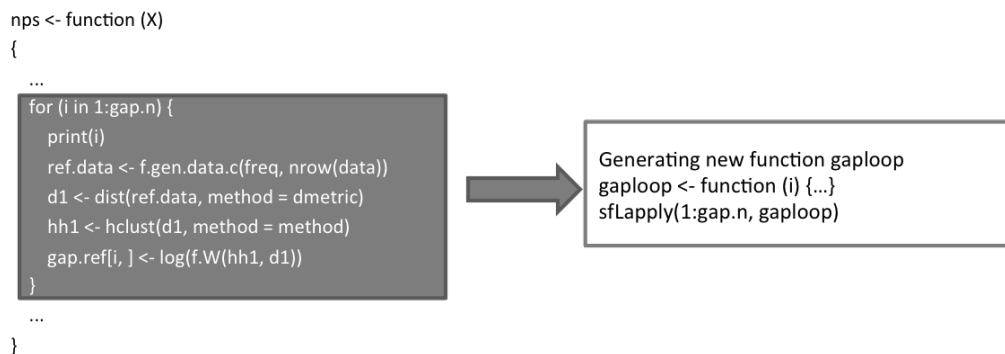


Figure 7.6: Parallelizing R `nps` function. The `grep` part on the left is the for loop code to be parallelized. The code within the brackets is node changed. The new function `gaploop` has a similar parameter `i`. This new function can be used by R parallel packages, such as Snow or Snowfall. The `sfLapply` is a function in the Snowfall package.

In this experiment parallel MPI function outperforms native R greatly with the subject number increasing. Especially in the last two bars the MPI function is more than 10 times

faster than native R.

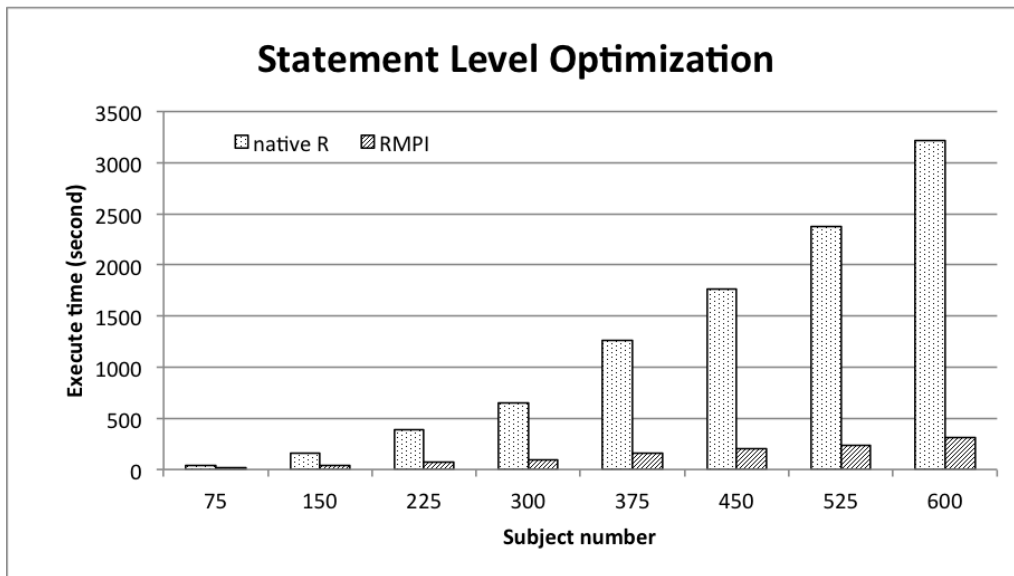


Figure 7.7: Statement level parallel method evaluation.

### 7.3.2 Data level parallelizing

The other method is data level parallelizing, usually used for large quantities of functions implemented by other programming language, such C, C++ and Fortran. It requires library re-compilation to modify such functions. Thus, data splitting is the most common way to parallelize such functions for most R users.

#### Design

This approach to applying MapReduce-like framework to calculating correlation matrices on gene expression data is inspired by work presented in [LKS10] who used a MapReduce-like model for distributing large datasets on GPUs to calculate Euclidean distance. In the case of this approach implementation, the input corresponds to a set of vectors, each containing the probesets intensity values for a single subject. This input

set of vectors is divided into data blocks that are dispatched to different mappers, dependent on the total number of CPU cores available. In the mapper stage, each data block is copied to each reducer by emitting as a key the index of the corresponding reducer and as the value the data block. Each reducer then calculates correlation values by loading the data block corresponding to the key and calculating the correlation coefficients against each data block output from the mappers.

In the example shown in Figure 1, each data block can contain several subjects, where in this example two subjects are showed per data block. Each block is loaded by a mapper, and then copied to each reducer. Based on the reducer ID, a corresponding data block is loaded and coefficients calculated against all other data blocks. For example, Reducer1 loads Data block 1 and pairwise it with outputs from each Mapper (Data block 1, Data block 2, and Data block 3), producing coefficient values  $d_{11}$ ,  $d_{12}$  and  $d_{13}$ .

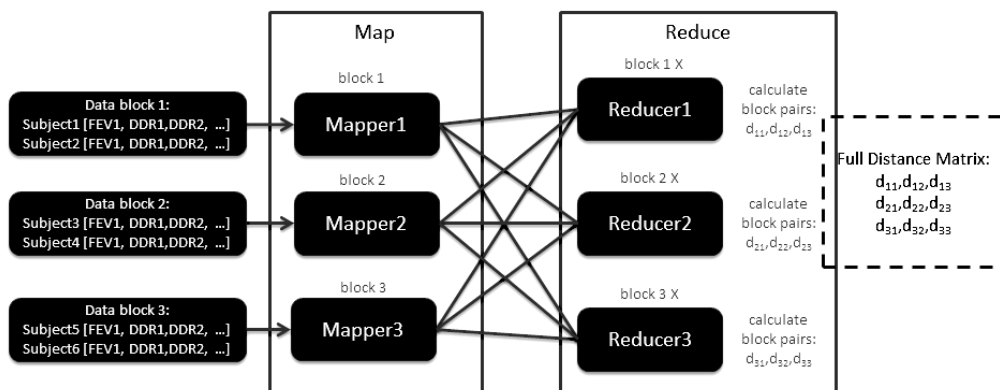


Figure 7.8: Basic correlation matrix calculation using MapReduce

To further optimize this MapReduce model, further modifications are made to the one presented by [LKS10]. There is a certain amount of redundancy in the calculation of a full correlation matrix. Each coefficient appears twice in the correlation matrix. Looking back at the example in Figure 7.8, each of  $d_{12}$  and  $d_{21}$ ,  $d_{13}$  and  $d_{31}$ , and  $d_{32}$  and  $d_{23}$  correspond to the same pair-wise calculations. To avoid this, the ID of each mapper can be compared with that of the target reducer before distributing data. If the Data block ID is greater than the reducer ID, then that particular data block distribution and correlation

coefficient calculation can then be skipped. For example, in Figure 2 mapper 2 does not send Data block 2 to reducer 1. This results in only  $d_{12}$  being calculated, instead of both  $d_{12}$  and  $d_{22}$ . This optimization results in all correlation coefficients only being calculated once.

This optimization the coefficient calculations shown in Figure 7.9 on results, however, now produces an imbalanced workload on the reducers. Reducer 1 receives a workload corresponding to one pair-wise calculation ( $d_{11}$ ), while Reducer 2 pairs calculations ( $d_{21}$ ,  $d_{22}$ ), and so forth. With this pattern of workload, the longest running reducer determines the overall performance of the MapReduce workflow. If the load can be balanced on the reducers, the workload will execute fully in parallel, thus reducing the overall execution time.

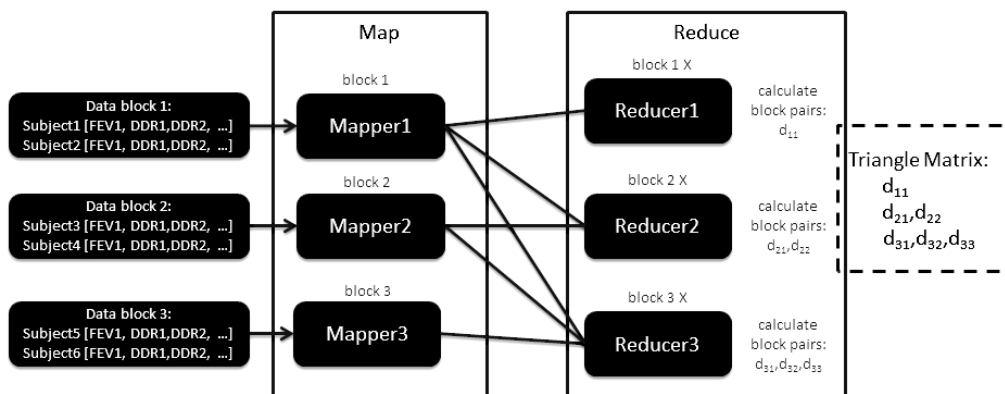


Figure 7.9: Correlation matrix calculation with redundant coefficient calculations skipped

To do this, a matrix transformation algorithm (see pseudo code below) has been created to balance all reducers by moving the bottom left triangular section of the correlation matrix to the top right, as shown in Figure 7.10.

Denote:

- $k$  is the number of all the data blocks.
- $i$  is the id of a data block.

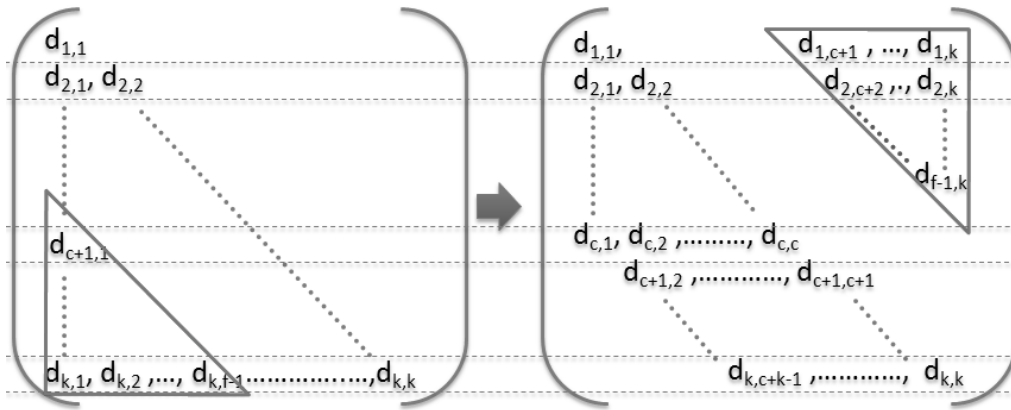


Figure 7.10: Matrix transformation. The elements in the bottom left triangle are mapped to the top right triangle. Each row represents a Reducers load, which contains either  $c$  elements or  $f$  elements.

- $a$  is the average number of distance calculation per reducer,  $a = \sum_{i=1}^k i/k$ .
- $c$  is the ceiling of  $a$ ,  $c = \lceil a \rceil$ .
- $f$  is the floor of  $a$ ,  $f = \lfloor a \rfloor$ .

Algorithm pseudo:

For the data block  $i$  in each *Mapper*

if ( $i \leq f$ )

*Mapper* send this block to *Reducer* <sub>$i$</sub> , *Reducer* <sub>$i+1$</sub> , ..., *Reducer* <sub>$i+f-1$</sub> ;

else if ( $i \geq c$ )

*Mapper* send this block to *Reducer* <sub>$i\%k$</sub> , *Reducer* <sub>$(i+1)\%k$</sub> , ..., *Reducer* <sub>$(i+c-1)\%k$</sub> ;

By using this matrix transformation to balance the reducers, each reducer will process either  $c$  pairs or  $f$  pairs of data blocks, where in theory all reducers load are fully balanced. In the example shown in Figure 7.9, six pairs of data blocks are calculated with an imbalanced workload. After balancing using the above matrix transformation, in Figure 7.11 each reducer now only calculates two pairs of data block.

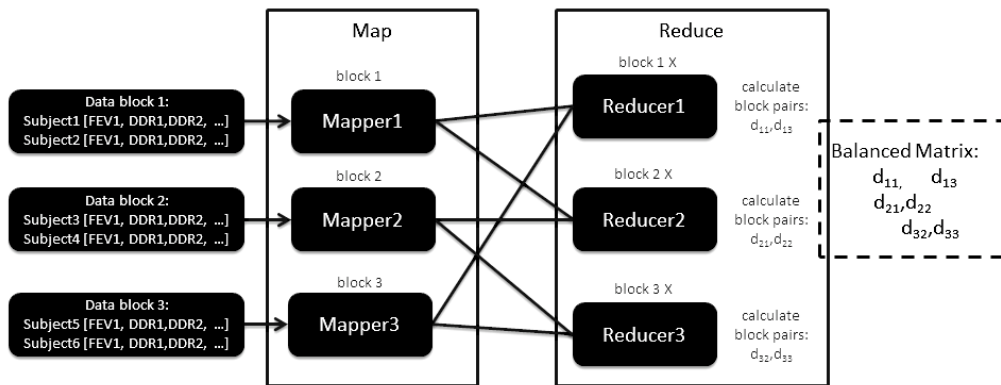


Figure 7.11: Balanced reducers.

Finally the workflow takes into account uneven numbers of available reducers to mappers. The hash function in MapReduce cannot always map mappers output to the expected number of reducers even if the mapper output keys are well designed. For example, if six mapper output keys are sent to six respective reducers, and only three reducers are available to receive the data, this results in one or more reducer receiving a greater workload to process sequentially. To ensure all reducers are fully utilized, calculations in reducers are shifted to combiners, which read pairwise data blocks directly from the file system, that calculate the result at the mapper stage before the hash mapping, as shown in Figure 7.12.

## Performance Evaluation

**Test datasets** This parallelizing method was tested against large datasets to validate being able to handle large studies that would be expected to applied in tranSMART. Three publicly available datasets: ONCOLOGY (GSE2109) [YMO<sup>+</sup>10] taken from NCBI GEO consisting on 2158 subjects and 54,675 probesets (1.9 GB comma-delimited value file), LEukemia (GSE13159) [KKR<sup>+</sup>08, HKW<sup>+</sup>10] consisting on 2096 subjects and 54,675 probesets (1.8 GB comma-delimited value file) MULTMYEL consisting on 559 subjects and 54,675 probesets (493 MB comma-delimited value file), and a breast invasive carcinoma dataset taken from TCGA [Net12] consisting of 547 subjects and 17,814 probesets

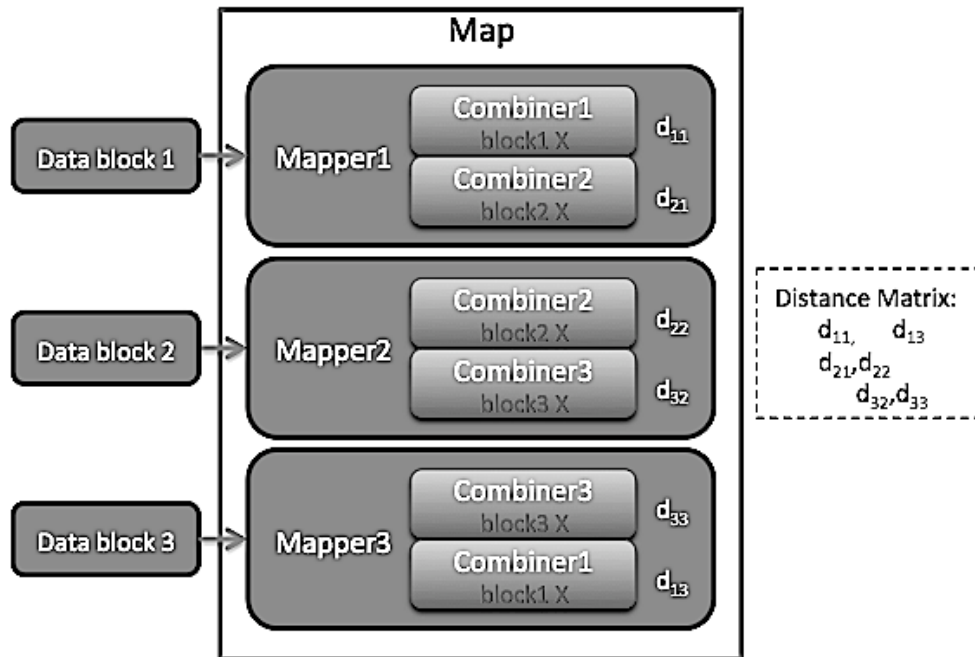


Figure 7.12: Calculation in Combiner. This is the MapReduce framework where all calculations in Reducers are moved to Combiner according to the algorithm in Figure 7.11.

(157 MB comma-delimited value file). OpenStack, a computational testbed, has been set up at Imperial College London for comparable virtual machine configurations for the R implementation and the MapReduce implementation.

In order to verify the parallelizing method all types of correlation functions were tested. Currently, there are three main types of correlation coefficient calculation algorithms, productmoment coefficient, rank correlation coefficients and other dependence measurements. Pearson correlation is used for productmoment type, Spearman and Kendall for rank type and Euclidean distance for other dependence measures, which are implemented in R packages, r-base and fields. A vanilla R instance is deployed to test the performance against parallel R (Snowfall) and MapReduce via RHIPE.

**Benchmarks** Two benchmarks are used for the performance evaluation.

The micro-benchmark used datasets MULTMYEL (559 subjects) [WPE<sup>+</sup>14]. The vanilla

R configuration used 32 CPU cores and 32GB of memory in a single VM. Snowfall used 4 VMs each with 8 CPU cores, 8GB of memory and 6 workers. RHIPE used 4 VMs each with 8 CPU cores, 8GB of memory and 6 Mappers.

The macro-benchmark used datasets ONCOLOGY, LEukemia and TCGA. Three main tests are performed using three datasets, including ONCOLOGY (2158 subjects), a cross-study consisting of ONCOLOGY and LEukemia (4254 subjects), and an artificial dataset consisting of dual ONCOLOGY and dual LEukemia (8508 subjects). Due to the extremely long execution time of Kendall correlation, only the smallest TCGA data was used to calculate Kendall correlation. The vanilla R configuration used 64 CPU cores and 64GB of memory in a single VM. The master node of Snowfall used 8 CPU cores, 8GB of memory and 6 workers, with 14 slave VMs each with 4 CPU cores, 4 GB of memory and 3 workers. The master node of RHIPE used 8 CPU cores and 8GB of memory using 6 Mappers, with 14 slave VMs each with 4 CPU cores and 4 GB of memory using 42 Mappers.

Each experiment consists of two stages: data preparation and calculation. There are five methods for comparison: vanilla R, default Snowfall using socket connections, optimised Snowfall using the Rmpi package, and RHIPE using both the basic MapReduce algorithm and the optimised one.

The vanilla R data preparation is loading comma-separated value (CSV) data matrix from an Ext4 (fourth extended filesystem) local file system to an R matrix object. With default Snowfall, the data preparation overhead consists of loading the CSV data matrix from an Ext4 local file system, initializing the worker nodes, and exporting the data matrix and code to the workers. The data matrix is split by row (subject) rather than data block, where the corresponding computation calculates the correlation between rows. With Snowfall using Rmpi, the data preparation overhead includes splitting the data matrix using the Linux split command and copying each of the data block files to every VM. The number



of data blocks depends on the number of workers.

During the calculations, Rmpi workers perform the same tasks as in the Mappers in MapReduce. Each worker loads each corresponding data block sequentially from the local Ext4 filesystem. After each data block is loaded, the worker performs the corresponding calculations. Using RHIPE, the data preparation overhead consists of splitting the data matrix and uploading each file to HDFS. The calculation then follows the algorithms described in the method section.

**Results** A performance evaluation was carried out between vanilla R, parallel R with Snowfall, and MapReduce implementation. A subject correlation was calculated on the all subjects, calculating the coefficient matrices of the two benchmarks using Euclidean distance, Pearson and Spearman correlation functions.

In the micro-benchmark, as shown in Figures 7.13, vanilla R performs fastest and default Snowfall performs the slowest. Vanilla R has a larger data preparation overhead than RHIPE, but the calculation itself greatly outperforms all the other methods. All parallel R methods do not perform any better due to the data communication overhead. There is an extreme example in default Snowfall Spearman where the parallel calculation is 9 times slower than vanilla R.

However, this result only considers loading data from a data matrix stored in a local disk. The actual data in current tranSMART are stored in databases. Data retrieval time should be taken into account. As shown in Figure 4.8, if relational model are used to store this microarray data, another three hundred seconds are added to the whole workflow. All vanilla R tests will become at least 7 times longer than the current results, as shown in Figure 7.14. Similar affection will happen to the optimized parallel R methods. Compared to the calculation time, data preparation time occupies much larger proportion. If the Microarray Key Value data model is used for this application, the vanilla R execution

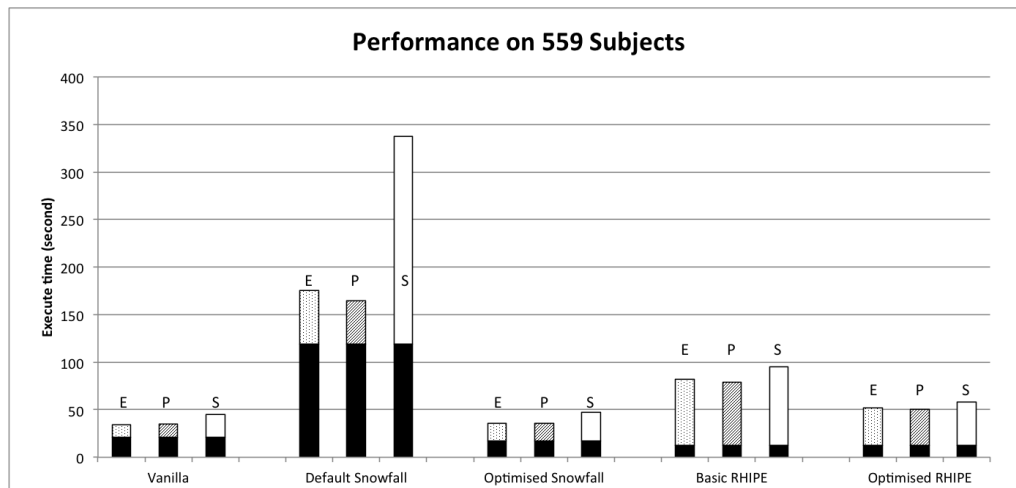


Figure 7.13: Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method show the data preparation time. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time.

time will be doubled only, as shown in Figure 7.15.

Though the vanilla R outperformed parallel R methods in the micro-benchmark, vanilla R may not scale very well due to its single data flow and single execution flow. Thus, the macro-benchmark was utilized to further test the optimised parallel R methods against vanilla R.

In the tests using the macro-benchmark, as shown in Figure 7.16, 7.17 and 7.18, though the optimised RHIPE outperforms all other methods, the optimised RHIPE calculation time is longer than optimised Snowfall. The only advantage using RHIPE is the faster data transfer via HDFS. In Figure 7.16 (2158 subjects), benefiting from the 3.30 times faster data preparation, the optimised RHIPE performs 1.22 - 1.30 times faster than the optimised Snowfall. In Figure 7.17 (4254 subjects), benefiting from the 3.69 times faster data preparation, the optimised RHIPE performs 1.31 - 1.49 times faster than the optimised Snowfall. In Figure 7.18 (8508 subjects), benefiting from the 5.13 times faster

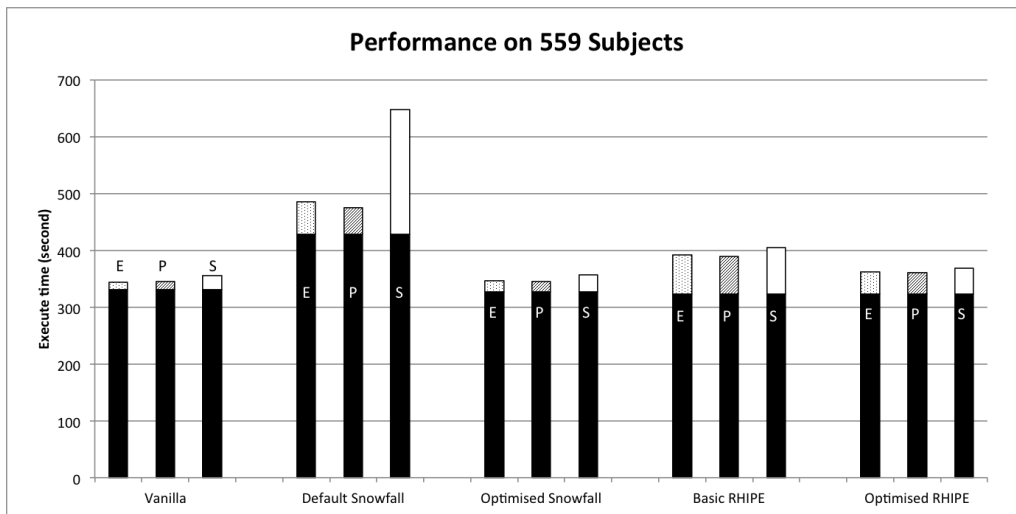


Figure 7.14: Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method show the data preparation time, including data retrieval from a relation database. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time.

data preparation, the optimised RHIPE performs 1.50 - 1.71 times faster than the optimised Snowfall and 7.24-16.56 times faster than the vanilla R. If multi-thread database retrieval functions are used for Snowfall data preparation, Snowfall can perform better and outperform RHIPE.

Therefore, Key Value data models hold great promise for large data analysis with the data size increasing. Other types of data, such as SNP or alignment, can be changed into data matrix before any calculation is performed. All such data can fully take advantage of these Key Value models.

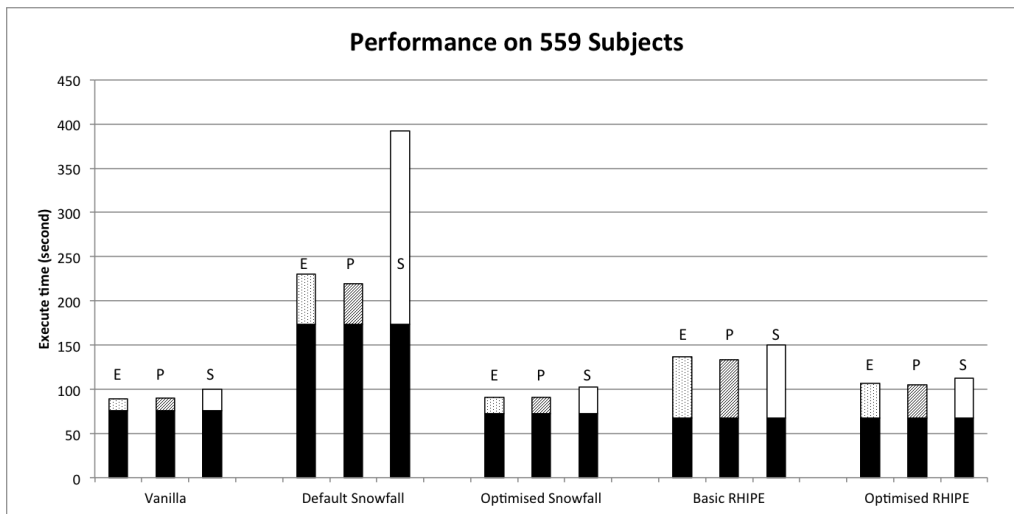


Figure 7.15: Performance on the micro-benchmark. This is the performance evaluation using vanilla R, default Snowfall package with socket connection, optimised Snowfall with Rmpi package and RHIPE package with the basic MapReduce algorithm and the optimised one. The bottom three bars in each method show the data preparation time, including data retrieval from a Key Value database. The vanilla R data preparation indicates loading data from a local file into memory; while in all parallel R methods, data copy almost occupy the whole data preparation time. The upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The bottom three bars indicate the data preparation time.

## 7.4 Conclusion

This chapter uses an analysis application the hierarchical clustering algorithm in transMART to show how the Key Value model affects the whole analysis workflow. An optimization method is also provided for correlation calculations used by the hierarchical clustering. The experiments in the performance evaluation section indicate the optimised RHIPE can perform up to 16.56 times faster than vanilla R, while Snowfall can conduct up to 9.68 times faster than vanilla R. Both the parallel R finished the long parallel Kendall correlation with all subjects of TCGA within 7 hours and conducted about 30 times faster than the estimated vanilla R. Compared to the traditional relational model, the optimized Rhipe method using the Key Value model performs more than 7 times faster than the same method using the relational model implemented in MySQL Cluster.

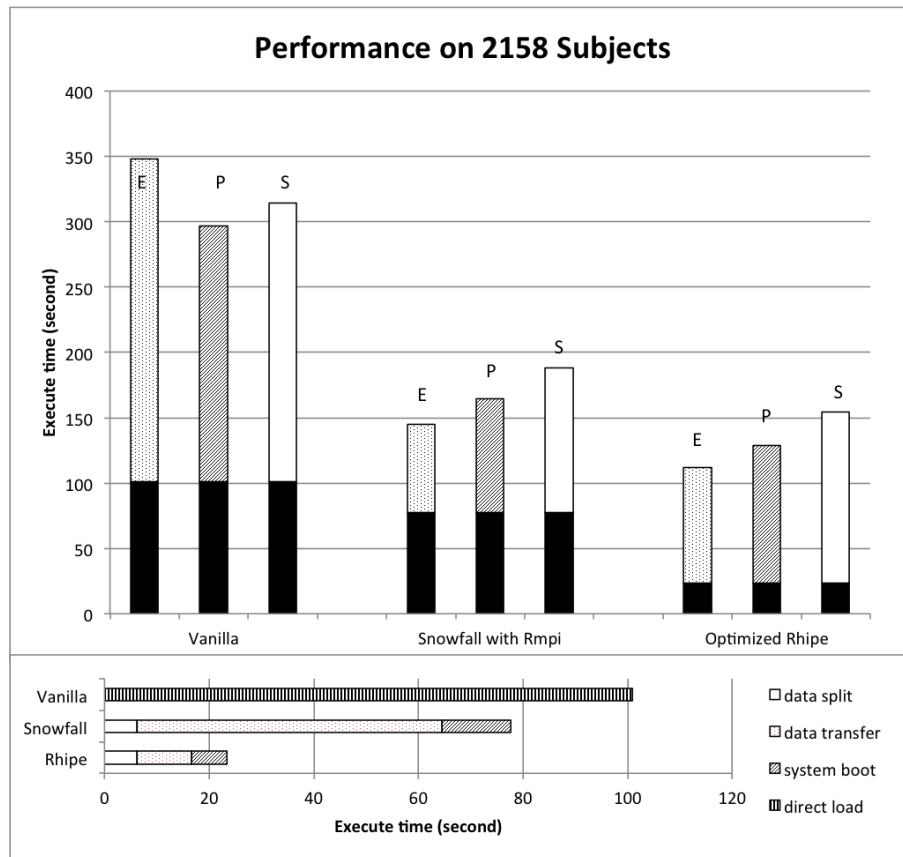


Figure 7.16: This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method show the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on ONCOLOGY dataset.

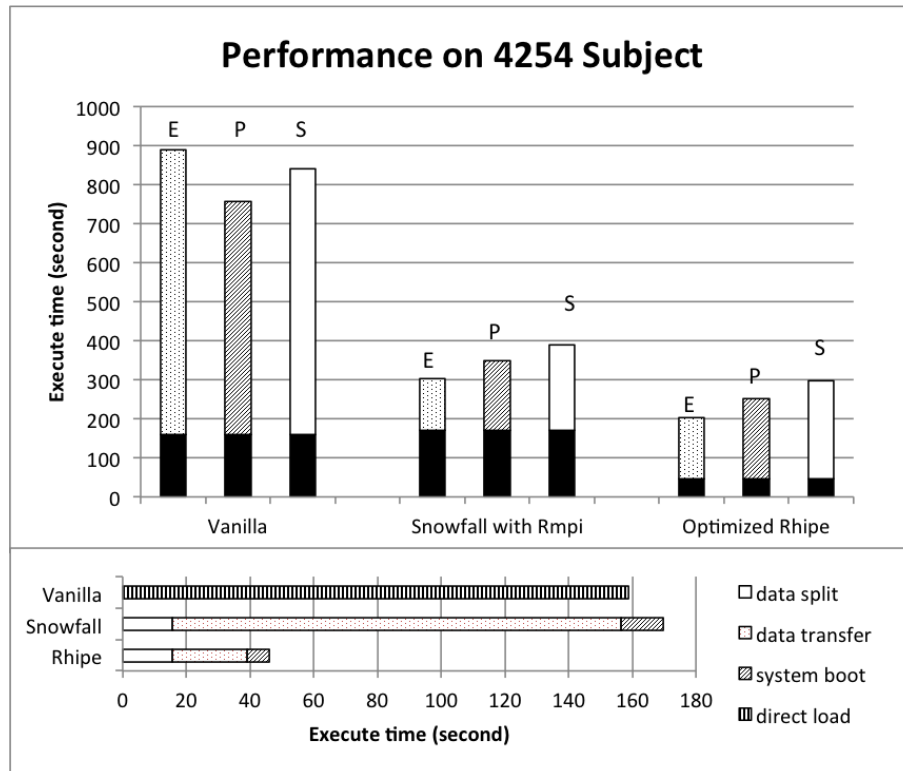


Figure 7.17: This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method show the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on the cross-study consisting of ONCOLOGY and LEukemia (4254 subjects).

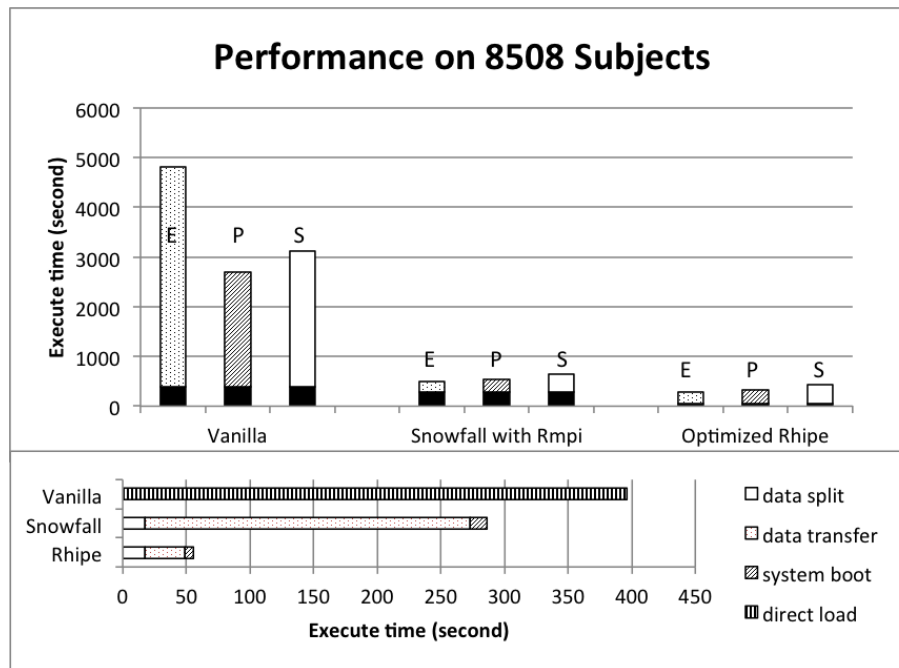


Figure 7.18: This is the performance evaluation using vanilla R, optimised Snowfall and optimised RHIPE package. The upper part of each figure indicates the total execution time. In this part, the bottom three bars in each method show the data preparation time; while the upper three bars respectively indicate the Euclidean (E), Pearson (P) and Spearman (S) calculation time. The lower part of each figure details the data preparation of each method. In this part, data split shows the time used for splitting the large data matrix into smaller pieces, data transfer for the Snowfall shows data copy time for the pieces to corresponding MPI workers, data transfer for the RHIPE shows the data uploading time for the same pieces to HDFS, system boot respectively shows the boot time of the MPI cluster and the Hadoop cluster, and the direct load shows the data loading time for vanilla R. Performance on the large artificial dataset (8508 subjects).

# Chapter 8

## Conclusion

### 8.1 Summary of Thesis Achievements

#### 8.1.1 Generic -omics big data model

A generic omics Key Value data model has been created for molecular profiling data. Three instances, including microarray data model, SNP data model and raw sequencing data model have been implemented and fully evaluated. Finally, an application demonstrates how Key Value models affect the performance.

#### 8.1.2 Microarray Key Value model

A Key Value data model for microarray data is created and compared to the relational model currently implemented in HBase, MySQL Cluster and MongoDB. The performance evaluation experiments shows the Key Value data model implemented in HBase outperformed the relational model implemented in HBase, MySQL Cluster and MongoDB by up to 246, 7 and 20 times. When this Key Value implementation integrated into



tranSMART, it performs up to 7 times faster than the current tranSMART.

### **8.1.3 SNP Key Value model**

The SNP Key Value model is aimed at solving the performance issues faced in translational research data storage databases, due to the increasing volume of data being produced. In the performance evaluation tests, the Key Value model in HBase outperforms the relational model on HBase and MySQL Cluster by up to 9-fold and 351-fold speedup respectively. When integrated into tranSMART this Key Value implementation performs up to 7 times faster than the current tranSMART.

### **8.1.4 Raw Sequencing Key Value model**

The raw sequencing Key Value data model is designed for queries over genotype and phenotype data, compared to the SAM data format implemented in HBase and a BAM server. In the performance evaluation experiments, the Key Value data model performs up to 440 and 22 times faster than the SAM format implemented in HBase and the BAM server.

### **8.1.5 Gene expression based patient stratification application**

The hierarchical clustering algorithm in tranSMART has been used as an analysis application to show how the Key Value model affects the whole analysis workflow. An optimization method is also provided for correlation calculations in the hierarchical clustering. Compared to the traditional relational model, the optimized method using the Key Value model performs more than 7 times faster than the same method using the relational model implemented in MySQL Cluster.

## 8.2 Future Work

### 8.2.1 Microarray Model

There are two query functions in HBase, Scan and Get. The current queries in the Key Value data model are optimized manually by choosing to perform either a pure Scan or pure Random Read operation. If a query optimizer built into the HBase system to automatically generate query plans, a mixed dynamically selected query method may perform better than choosing one or the other.

### 8.2.2 SNP Model

For queries by subject using the Subject table, the general SNP information are not stored together with genotypes sharing the same Row Key. An extra random read operation is required to read the general SNP information. The worst case is only one expected subject in a dataset. Then the subject data loading time may be doubled due to the extra search for SNP information.

### 8.2.3 Raw Sequencing Model

#### Extra disk space

The Key Value model is a space for time approach and consumes large disk space. For example, in the performance evaluation section of Chapter 6, the BAM file size is 1.29 GB (the corresponding SAM is 5.69 GB), but HBase used 29.16 GB to store three copies of the data. Each used 7.53 times of storage space of BAM. The main reason for this large overhead is the column number. If a row contains N columns, the Row Key will

be duplicated  $N-1$  times. Further the composite Row Key in the Key Value model contains multiple columns, which lead to a even larger overhead. A possible way to reduce this overhead is to merge columns. If some columns are frequently used together, such RNEXT and PNEXT, it is possible to reduce one Row Key space by merging column RNEXT and column PNEXT into column RNEXT+PNEXT. When consider Family together, if a Family becomes smaller, query over the Family will be much faster than a larger one. But when many Families are searched concurrently, the combination of smaller Families may be even slower than the larger one. Also, new Families will lead to extra copies of Row Key.

### **Long MML binning schema**

In the most popular NGS sequencing machines, such as Illumina [Gen10] and SOLiD [Dek07], the length of a RNA-Seq read is usually less than 300 bp. For example, the raw sequencing dataset HG00096CH11 published in 1000 Genomes only contains reads mapped to less than 200 bases. But if long mapped reads are emerging, the scan method used in Chapter 6 may not be suitable. The whole dataset can be divided into two bins: one containing reads whose MML is shorter than a certain number CLEN and the other one containing all remaining raw sequences, including raw sequences with MML not shorter than CLEN bases. Then a Long MML schema is required.

### **8.2.4 Coprocessors**

HBase has very effective MapReduce integration for distributed computation over data stored within its tables, but pushing the computation up to the server where the data can be operated directly without communication overheads can give a dramatic performance improvement over HBases already scanning performance.

Before HBase-0.92, it was not possible to extend HBase with custom functionality except by extending the base classes. But new Coprocessors feature of HBase [VS11], a framework for both flexible and generic extension, and of distributed computation directly within the HBase server processes. The idea of HBase Coprocessors was inspired by Google's BigTable coprocessors. Thus, many calculations, such as correlations and tests [H<sup>+</sup>51, Law38, SB01], can be done in the server side without communication overheads.

# Bibliography

- [ABHG13] Brian D Athey, Michael Braxenthaler, Magali Haas, and Yike Guo. transMART: An Open Source and Community-Driven Informatics and Data Sharing Platform for Clinical and Translational Research. *AMIA Summits on Translational Science proceedings AMIA Summit on Translational Science*, 2013:6–8, 2013.
- [AD11] Maurizio Atzori and Nicoletta Dessì. Dataspace: Where structure and schema meet. In *Learning Structure and Schemas from Documents*, pages 97–119. Springer, 2011.
- [Ade07] Clement A Adebamowo. International haplotype mapping project. *Annals of Ibadan Postgraduate Medicine*, 2(2):8–11, 2007.
- [AGM+90] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
- [Ahl96] Christopher Ahlberg. Spotfire: an information exploration environment. *ACM SIGMOD Record*, 25(4):25–29, 1996.
- [AL05] Maha Abdallah and Hung Cuong Le. Scalable range query processing for large-scale distributed database applications. In *IASTED PDCS*, pages 433–439, 2005.

- [ALA10] DM Altshuler, ES Lander, and L Ambrogio. A map of human genome variation from population scale sequencing. *Nature*, 476:1061–1073, 2010.
- [ALS10] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide*. ” O’Reilly Media, Inc.”, 2010.
- [AMC<sup>+</sup>13] Simon Anders, Davis J McCarthy, Yunshun Chen, Michal Okoniewski, Gordon K Smyth, Wolfgang Huber, and Mark D Robinson. Count-based differential expression analysis of RNA sequencing data using R and Bioconductor. *Nature protocols*, 8:1765–86, 2013.
- [And10] Simon Anders. Analysing rna-seq data with the deseq package. *Molecular biology*, pages 1–17, 2010.
- [AOVP08] Jeannie Albrecht, David Oppenheimer, Amin Vahdat, and David A Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Transactions on Internet Technology (TOIT)*, 8(4):18, 2008.
- [APH14] S. Anders, P. T. Pyl, and W. Huber. HTSeq A Python framework to work with high-throughput sequencing data. Technical report, 2014.
- [ARH12] Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from rna-seq data. *Genome research*, 22(10):2008–2017, 2012.
- [BBBB95] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.
- [BBK01] Christian Böhm, Stefan Berchtold, and Daniel A Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

- [BDDY09] Pete Belknap, Benoit Dageville, Karl Dias, and Khaled Yagoub. Self-tuning for SQL performance in oracle database 11g. In *Proceedings - International Conference on Data Engineering*, pages 1694–1700, 2009.
- [BE77] Mike W. Blasgen and Kapali P. Eswaran. Storage and access in relational data bases. *IBM Systems Journal*, 16(4):363–377, 1977.
- [BFT<sup>+</sup>05] Michael Barnes, Johannes Freudenberg, Susan Thompson, Bruce Aronow, and Paul Pavlidis. Experimental comparison and cross-validation of the affymetrix and illumina gene expression analysis platforms. *Nucleic acids research*, 33(18):5914–5923, 2005.
- [BGQ<sup>+</sup>11] Derek W Barnett, Erik K Garrison, Aaron R Quinlan, Michael P Strömberg, and Gabor T Marth. Bamtools: a c++ api and toolkit for analyzing and managing bam files. *Bioinformatics*, 27(12):1691–1692, 2011.
- [big08] Bigtable : A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 26:1–26, 2008.
- [BM72] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes, 1972.
- [BM02] Rudolf Bayer and Edward McCreight. *Organization and maintenance of large ordered indexes*. Springer, 2002.
- [Bor08] Dhruva Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT* [http://hadoop.apache.org/common/docs/current/hdfs\\_design.pdf](http://hadoop.apache.org/common/docs/current/hdfs_design.pdf), 2008.
- [Bro12] Martin C Brown. *Getting Started with Couchbase Server*. ” O’Reilly Media, Inc.”, 2012.
- [BWL<sup>+</sup>13] Tanya Barrett, Stephen E Wilhite, Pierre Ledoux, Carlos Evangelista, Irene F Kim, Maxim Tomashevsky, Kimberly a Marshall, Katherine H

- Phillippy, Patti M Sherman, Michelle Holko, Andrey Yefanov, Hyeseung Lee, Naigong Zhang, Cynthia L Robertson, Nadezhda Serova, Sean Davis, and Alexandra Soboleva. NCBI GEO: archive for functional genomics data sets—update. *Nucleic acids research*, 41(Database issue):D991–5, January 2013.
- [Car58] A Carlisle. A guide to the named variants of scots pine (*pinus silvestris* linnaeus). *Forestry*, 31(2):203–224, 1958.
- [Car13] Josiah L Carlson. *Redis in Action*. Manning Publications Co., 2013.
- [Cat11] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
- [CBW<sup>+</sup>97] J Michael Cherry, Catherine Ball, Shuai Weng, Gail Juvik, Rita Schmidt, Caroline Adler, Barbara Dunn, Selina Dwight, Linda Riles, Robert K Mortimer, et al. Genetic and physical maps of *saccharomyces cerevisiae*. *Nature*, 387(6632 Suppl):67, 1997.
- [CCF<sup>+</sup>08] Michael Cafarella, Edward Chang, Andrew Fikes, Alon Halevy, Wilson Hsieh, Alberto Lerner, Jayant Madhavan, and S. Muthukrishnan. Data management projects at Google, 2008.
- [CFCS04] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2(1):3–14, 2004.
- [Cha83] Tse-Wen Chang. Binding of cells to matrixes of distinct antibodies coated on solid surface. *Journal of immunological methods*, 65(1):217–223, 1983.
- [Cho13] Kristina Chodorow. *MongoDB: the definitive guide*. ” O’Reilly Media, Inc.”, 2013.



- [CL12] Michael Cariaso and Greg Lennon. SNPedia: A wiki supporting personal genome annotation, interpretation and analysis. *Nucleic Acids Research*, 40, 2012.
- [CLH<sup>+</sup>99] G Cipriani, G Lot, W-G Huang, MT Marrazzo, E Peterlunger, and R Testolin. Ac/gt and ag/ct microsatellite repeats in peach [*Prunus persica* (L) Batsch]: isolation, characterisation and cross-species amplification in *Prunus*. *Theoretical and Applied Genetics*, 99(1-2):65–72, 1999.
- [Com79] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2):121–137, 1979.
- [Con07] The International HapMap Consortium. A second generation human haplotype map of over 3.1 million SNPs. *Nature*, 449:851–61, 2007.
- [COS<sup>+</sup>12] Wei-Chen Chen, George Ostouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu. A quick guide for the pbdmpi package. *R Vignette*, URL <http://cran.r-project.org/package=pbdMPI>, 2012.
- [Cro06] Douglas Crockford. Json: The fat-free alternative to xml. In *Proc. of XML*, volume 2006, 2006.
- [CRS<sup>+</sup>08] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!’s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.
- [CSK<sup>+</sup>03] Ramu Chenna, Hideaki Sugawara, Tadashi Koike, Rodrigo Lopez, Toby J Gibson, Desmond G Higgins, and Julie D Thompson. Multiple sequence alignment with the clustal series of programs. *Nucleic acids research*, 31(13):3497–3500, 2003.

- [CVFB03] Chris Cheadle, Marquis P Vawter, William J Freed, and Kevin G Becker. Analysis of microarray data using Z score transformation. *The Journal of molecular diagnostics : JMD*, 5:73–81, 2003.
- [DAA<sup>+</sup>11] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, and Richard Durbin. The variant call format and VCFtools. *Bioinformatics*, 27:2156–2158, 2011.
- [Dek07] Cees Dekker. Solid-state nanopores. *Nature nanotechnology*, 2:209–215, 2007.
- [Del00] Kalen Delaney. *Inside Microsoft SQL Server 2000*. Microsoft Press, 2000.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51:107, 2008.
- [DGG<sup>+</sup>86] David J DeWitt, Robert H Gerber, Goetz Graefe, M Heytens, K Kumar, and M Muralikrishna. *A High Performance Dataflow Database Machine*. Computer Science Department, University of Wisconsin, 1986.
- [ECS<sup>+</sup>08] George Eadon, Eugene Inseok Chong, Shrikanth Shankar, Ananth Raghavan, Jagannathan Srinivasan, and Souripriya Das. Supporting table partitioning by reference in Oracle. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1111–1122, 2008.
- [FKT86] Shinya Fushimi, Masaru Kitsuregawa, and Hidehiko Tanaka. An overview of the system software of a parallel relational database machine grace. In *VLDB*, volume 86, pages 209–219, 1986.
- [FLW03] Martin C Frith, Michael C Li, and Zhiping Weng. Cluster-buster: Finding dense clusters of motifs in dna sequences. *Nucleic acids research*, 31(13):3666–3668, 2003.

- [FYL<sup>+</sup>15] Chen Feng, Xi Yang, Fan Liang, Xian-He Sun, and Zhiwei Xu. Lcindex: A local and clustering index on distributed ordered tables for flexible multi-dimensional range queries. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 719–728. IEEE, 2015.
- [Gar05] Howard N Garb. Clinical judgment and decision making. *Annual review of clinical psychology*, 1:67–89, 2005.
- [GCB<sup>+</sup>04] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.
- [GCK<sup>+</sup>10] Andrey Gusev, Jonathan Creighton, Raj K Kammend, Sarat Kakarla, and David Brower. Special values in oracle clusterware resource profiles, January 15 2010. US Patent App. 12/688,710.
- [Gen10] Cluster Generation. Illumina Sequencing Technology. *Image Rochester NY*, 21:1–5, 2010.
- [Geo08] Lars George. *HBase The Definitive Guide*. 2008.
- [GGT10] Li Guo, Yike Guo, and Xiangchuan Tian. IC Cloud: A Design Space for Composable Cloud Computing. *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010.
- [GHR12a] Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from rna-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.
- [GHR<sup>+</sup>12b] Saptarshi Guha, Ryan Hafen, Jeremiah Rounds, Jin Xia, Jianfu Li, Bowei Xi, and William S Cleveland. Large complex data: divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012.

- [Gil03] Don Gilbert. Sequence file format conversion with command-line readseq. *Current Protocols in Bioinformatics*, pages A–1E, 2003.
- [GMPZ<sup>+</sup>08] Elizabeth Garrett-Mayer, Giovanni Parmigiani, Xiaogang Zhong, Leslie Cope, and Edward Gabrielson. Cross-study validation and combined analysis of gene expression microarray data. *Biostatistics (Oxford, England)*, 9:333–354, 2008.
- [GPN<sup>+</sup>10] Naohisa Goto, Pjotr Prins, Mitsuteru Nakao, Raoul Bonnal, Jan Aerts, and Toshiaki Katayama. Bioruby: bioinformatics software for the ruby programming language. *Bioinformatics*, 26(20):2617–2619, 2010.
- [Gut84] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.
- [H<sup>+</sup>51] Harold Hotelling et al. A generalized t test and measure of multivariate dispersion. In *Proceedings of the second Berkeley symposium on mathematical statistics and probability*. The Regents of the University of California, 1951.
- [HC03] Xiaoqiu Huang and Kun-Mao Chao. A generalized global alignment algorithm. *Bioinformatics*, 19(2):228–233, 2003.
- [HC10] Margaret A Hamburg and Francis S Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.
- [HHZ<sup>+</sup>06] I Hanamura, Y Huang, F Zhan, B Barlogie, and J Shaughnessy. Prognostic value of cyclin D2 mRNA expression in newly diagnosed multiple myeloma treated with high-dose chemotherapy and tandem autologous stem cell transplantations. *Leukemia*, 20(7):1288–90, July 2006.
- [HJCT88] William E Holben, Janet K Jansson, Barry K Chelm, and James M Tiedje. Dna probe method for the detection of specific microorganisms in the soil

- bacterial community. *Applied and Environmental Microbiology*, 54(3):703–711, 1988.
- [HK10] Thomas J Hardcastle and Krystyna A Kelly. bayseq: empirical bayesian methods for identifying differential expression in sequence count data. *BMC bioinformatics*, 11(1):422, 2010.
- [HKJR10] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.
- [HKW<sup>+</sup>10] Torsten Haferlach, Alexander Kohlmann, Lothar Wieczorek, Giuseppe Basso, Geertruy Te Kronnie, Marie-Christine Béné, John De Vos, Jesus M Hernández, Wolf-Karsten Hofmann, Ken I Mills, et al. Clinical utility of microarray-based gene expression profiling in the diagnosis and subclassification of leukemia: report from the international microarray innovations in leukemia study group. *Journal of Clinical Oncology*, 28(15):2529–2537, 2010.
- [IFY<sup>+</sup>93] Chikashi Ishioka, Thierry Frebourg, Yu-Xin Yan, Marc Vidal, Stephen H Friend, Susanne Schmidt, and Richard Iggo. Screening patients for heterozygous p53 mutations using a functional assay in yeast. *Nature genetics*, 5(2):124–129, 1993.
- [KAB<sup>+</sup>12] Misha Kapushesky, Tomasz Adamusiak, Tony Burdett, Aedin Culhane, Anna Farne, Alexey Filippov, Ele Holloway, Andrey Klebanov, Nataliya Kryvych, Natalja Kurbatova, Pavel Kurnosov, James Malone, Olga Melnichuk, Robert Petryszak, Nikolay Pultsin, Gabriella Rustici, Andrew Tikhonov, Ravensara S Travillian, Eleanor Williams, Andrey Zorin, Helen Parkinson, and Alvis Brazma. Gene Expression Atlas update a value-added

- database of microarray and sequencing-based functional genomics experiments. *Gene Expression*, 40:1–5, 2012.
- [Ken48] Maurice George Kendall. Rank correlation methods. 1948.
- [KG06] Ankur Khetrapal and Vinay Ganesh. HBase and Hypertable for large scale distributed storage systems A Performance evaluation for Open Source BigTable Implementations. *Evaluation*, page 8, 2006.
- [KHK11] Donna Karolchik, Angie S. Hinrichs, and W. James Kent. The UCSC genome browser. *Current Protocols in Human Genetics*, 2011.
- [KKR<sup>+</sup>08] Alexander Kohlmann, Thomas J Kipps, Laura Z Rassenti, James R Downing, Sheila A Shurtleff, Ken I Mills, Amanda F Gilkes, Wolf-Karsten Hofmann, Giuseppe Basso, Marta Campo DellOrto, et al. An international standardization programme towards the application of gene expression profiling in routine leukaemia diagnostics: the microarray innovations in leukemia study prephase. *British journal of haematology*, 142(5):802–807, 2008.
- [KNV<sup>+</sup>06] Michael Krawczak, Susanna Nikolaus, Huberta Von Eberstein, Peter J P Croucher, Nour Eddine El Mokhtari, and Stefan Schreiber. PopGen: Population-based recruitment of patients and controls for the analysis of complex genotype-phenotype relationships. In *Community Genetics*, volume 9, pages 55–61, 2006.
- [KPB09] Jochen Knaus, Christine Porzelius, and Harald Binder. Easier Parallel Computing in R with snowfall and sfCluster. *Source*, 1:54–59, 2009.
- [KRB<sup>+</sup>89] Bat-sheva Kerem, Johanna M Rommens, Janet A Buchanan, Danuta Markiewicz, Tara K Cox, Aravinda Chakravarti, Manuel Buchwald, and Lap-Chee Tsui. Identification of the cystic fibrosis gene: genetic analysis. *Science*, 245(4922):1073–1080, 1989.

- [KS<sup>+</sup>98] Michael Krawczak, Jörg Schmidtke, et al. *DNA fingerprinting*. BIOS Scientific Publishers Ltd, 1998.
- [KSF<sup>+</sup>02] W James Kent, Charles W Sugnet, Terrence S Furey, Krishna M Roskin, Tom H Pringle, Alan M Zahler, and David Haussler. The human genome browser at ucsc. *Genome research*, 12(6):996–1006, 2002.
- [LaF09] Thomas LaFramboise. *Single nucleotide polymorphism arrays: A decade of biological, computational and technological advances*, 2009.
- [Law38] DN Lawley. A generalization of fisher’s z test. *Biometrika*, pages 180–187, 1938.
- [LHW<sup>+</sup>09] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25:2078–2079, 2009.
- [LKS10] Qi Li, Vojislav Kecman, and Raied Salman. A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 208–213. Ieee, December 2010.
- [LLB<sup>+</sup>01] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.
- [LLS<sup>+</sup>03] Guoying Liu, Ann E Loraine, Ron Shigeta, Melissa Cline, Jill Cheng, Venu Valmeekam, Shaw Sun, David Kulp, and Michael A Siani-Rose. Netaffx: Affymetrix probesets and annotations. *Nucleic acids research*, 31(1):82–86, 2003.

- [LM10] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44:35, 2010.
- [LNM<sup>+</sup>12] S Lund, Dan Nettleton, D McCarthy, G Smyth, et al. Detecting differential expression in rna-sequence data using quasi-likelihood with shrunken dispersion estimates. *Statistical applications in genetics and molecular biology*, 11(5):8, 2012.
- [LRPR98] E Lai, J Riley, I Purvis, and A Roses. A 4-Mb high-density single nucleotide polymorphism-based map around human APOE. *Genomics*, 54(1):31–8, November 1998.
- [LY01] Lee-Chin Hsu Liu and Kenji Yoneda. Secondary index search, 2001. US Patent 6,266,660.
- [LYM] Hao Lin, Shuo Yang, and Samuel P Midkiff. A parallel r framework for processing large dataset on distributed systems.
- [MC00] John E Mickle and Garry R Cutting. Genotype-phenotype relationships in cystic fibrosis. *Medical Clinics of North America*, 84(3):597–607, 2000.
- [Met10] Michael L Metzker. Sequencing technologies - the next generation. *Nature reviews. Genetics*, 11:31–46, 2010.
- [MHG10] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co., 2010.
- [MM01] David W Mount and David W Mount. *Bioinformatics: sequence and genome analysis*, volume 2. Cold spring harbor laboratory press New York:, 2001.



- [MMN<sup>+</sup>04] John MacCormick, Nick Murphy, Marc Najork, Chandramohan A Thekkath, and Lidong Zhou. Boxwood: Abstractions as the foundation for storage infrastructure. In *OSDI*, volume 4, pages 8–8, 2004.
- [Mom01] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.
- [MWM<sup>+</sup>08] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature methods*, 5(7):621–628, 2008.
- [MWM<sup>+</sup>10] Shawn N Murphy, Griffin Weber, Michael Mendis, Vivian Gainer, Henry C Chueh, Susanne Churchill, and Isaac Kohane. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). *Journal of the American Medical Informatics Association : JAMIA*, 17(2):124–30, 2010.
- [Net12] Cancer genome Atlas Reserach Network. Comprehensive molecular portraits of human breast tumours. *Nature*, 490:61–70, 2012.
- [NTT97] Deborah A Nickerson, Vincent O Tobe, and Scott L Taylor. Polyphred: automating the detection and genotyping of single nucleotide substitutions using fluorescence-based resequencing. *Nucleic acids research*, 25(14):2745–2751, 1997.
- [OCGO96] Patrick O’Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O’Neil. The log-structured merge-tree (LSM-tree), 1996.
- [OD14] Bogdan Oancea and Raluca Mariana Dragoescu. Integrating r and hadoop for big data analysis. *arXiv preprint arXiv:1407.4908*, 2014.

- [OMN10] Brian D O'Connor, Barry Merriman, and Stanley F Nelson. SeqWare Query Engine: storing and searching sequence data in the cloud. *BMC bioinformatics*, 11 Suppl 12:S2, 2010.
- [PCG<sup>+</sup>10] Vlad Popovici, Weijie Chen, Brandon G Gallas, Christos Hatzis, Weiwei Shi, Frank W Samuelson, Yuri Nikolsky, Marina Tsyganova, Alex Ishkin, Tatiana Nikolskaya, Kenneth R Hess, Vicente Valero, Daniel Booser, Mauro Delorenzi, Gabriel N Hortobagyi, Leming Shi, W Fraser Symmans, and Lajos Pusztai. Effect of training-sample size and classification difficulty on the accuracy of genomic predictors. *Breast cancer research : BCR*, 12:5, 2010.
- [Pet08] Jure Petrovic. Using memcached for data distribution in industrial environment. In *ICONS*, pages 368–372, 2008.
- [PG11] Swapnil Patil and Garth A Gibson. Scale and concurrency of giga+: File system directories with millions of files. In *FAST*, volume 11, pages 13–13, 2011.
- [PGL99] Raymond J. Peterson, David Goldman, and Jeffrey C. Long. Effects of worldwide population subdivision on ALDH2 linkage disequilibrium. *Genome Research*, 9:844–852, 1999.
- [Pop72] JG Pope. An investigation of the accuracy of virtual population analysis using cohort analysis. *ICNAF Research Bulletin*, 9(10):65–74, 1972.
- [PSH<sup>+</sup>98] Alexander Poltorak, Irina Smirnova, Xiaolong He, Mu-Ya Liu, Christophe Van Huffel, Dale Birdwell, Erica Alejos, Maria Silva, Xin Du, Patricia Thompson, et al. Genetic and physical mapping of the *i<sub>l</sub>1<sub>3</sub>* locus: Identification of the toll-4 receptor as a candidate gene in the critical region. *Blood Cells, Molecules, and Diseases*, 24(3):340–355, 1998.

- [Qua01] J Quackenbush. Computational analysis of microarray data. *Nature reviews. Genetics*, 2:418–427, 2001.
- [R C14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.
- [RBS02] Vasily Ramensky, Peer Bork, and Shamil Sunyaev. Human non-synonymous snps: server and survey. *Nucleic acids research*, 30(17):3894–3900, 2002.
- [RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.
- [RE13] Juris Rats and Gints Ernestsons. Clustering and ranked search for enterprise content management. *International Journal of E-Entrepreneurship and Innovation (IJEEI)*, 4(4):20–31, 2013.
- [RFH<sup>+</sup>01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.
- [RHAG<sup>+</sup>90] Brigitte Rigat, Christine Hubert, Francois Alhenc-Gelas, Franmois Cambien, Pierre Corvol, and Florent Soubrier. An insertion/deletion polymorphism in the angiotensin i-converting enzyme gene accounting for half the variance of serum enzyme levels. *Journal of Clinical Investigation*, 86(4):1343, 1990.
- [RHCS92] Brigitte Rigat, Christine Hubert, Pierre Corvol, and Florent Soubrier. Pcr detection of the insertion/deletion polymorphism of the human angiotensin converting enzyme gene (dcp1)(dipeptidyl carboxypeptidase 1). *Nucleic acids research*, 20(6):1433, 1992.

- [RIST<sup>+</sup>12] Caryn S Ross-Innes, Rory Stark, Andrew E Teschendorff, Kelly A Holmes, H Raza Ali, Mark J Dunning, Gordon D Brown, Ondrej Gojis, Ian O Ellis, Andrew R Green, et al. Differential oestrogen receptor binding is associated with clinical outcome in breast cancer. *Nature*, 481(7381):389–393, 2012.
- [RKP<sup>+</sup>96] Mostafa Ronaghi, Samer Karamohamed, Bertil Pettersson, Mathias Uhlén, and Pål Nyrén. Real-time dna sequencing using detection of pyrophosphate release. *Analytical biochemistry*, 242(1):84–89, 1996.
- [RLB<sup>+</sup>00] Peter Rice, Ian Longden, Alan Bleasby, et al. Emboss: the european molecular biology open software suite. *Trends in genetics*, 16(6):276–277, 2000.
- [RLG<sup>+</sup>06] Michael Reich, Ted Liefeld, Joshua Gould, Jim Lerner, Pablo Tamayo, and Jill P Mesirov. GenePattern 2.0., 2006.
- [RMS10] Mark D Robinson, Davis J McCarthy, and Gordon K Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, 2010.
- [RPB<sup>+</sup>09] Marc S Raab, Klaus Podar, Iris Breitkreutz, Paul G Richardson, and Kenneth C Anderson. Multiple myeloma. *Lancet*, 374:324–339, 2009.
- [RSS<sup>+</sup>12] Mark D Robinson, Dario Strbenac, Clare Stirzaker, Aaron L Statham, Jenny Z Song, Terence P Speed, and Susan J Clark. Copy-number-aware differential analysis of quantitative dna sequencing data. *Genome research*, pages gr–139055, 2012.
- [RT04] Mikael Ronstrom and Lars Thalmann. Mysql cluster architecture overview. *MySQL Technical White Paper*, 2004.
- [SB01] Albert Satorra and Peter M Bentler. A scaled difference chi-square test statistic for moment structure analysis. *Psychometrika*, 66(4):507–514, 2001.

- [SBAJ<sup>+</sup>09] Mouna Stayoussef, Jihen Benmansour, Fayza A Al-Jenaidi, Rita Nemr, Muhallab E Ali, Touhami Mahjoub, and Wassim Y Almawi. Influence of common and specific HLA-DRB1/DQB1 haplotypes on genetic susceptibilities of three distinct Arab populations to type 1 diabetes. *Clinical and vaccine immunology : CVI*, 16(1):136–8, January 2009.
- [SBB<sup>+</sup>02] Jason E Stajich, David Block, Kris Boulez, Steven E Brenner, Stephen A Chervitz, Chris Dagdigian, Georg Fuellen, James GR Gilbert, Ian Korf, Hilmar Lapp, et al. The bioperl toolkit: Perl modules for the life sciences. *Genome research*, 12(10):1611–1618, 2002.
- [SF05] Roland B Stoughton and Stephen H Friend. How molecular profiling could revolutionize drug discovery. *Nature reviews. Drug discovery*, 4:345–350, 2005.
- [SF09] Adam Silberstein and Rodrigo Fonseca. Adaptively Parallelizing Distributed Range Queries. *System*, 2:682–693, 2009.
- [SFG13] Ranjan Sen, Andrew Farris, and Peter Guerra. Benchmarking apache accumulo bigdata distributed table store using its continuous test suite. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 334–341. IEEE, 2013.
- [Siv08] Nayanah Siva. 1000 genomes project. *Nature biotechnology*, 26(3):256–256, 2008.
- [SKKP10] Sándor Szalma, Venkata Koka, Tatiana Khasanova, and Eric D Perakslis. Effective knowledge management in translational medicine. *Journal of translational medicine*, 8(1):68, 2010.

- [SL03] Jeffrey M Squyres and Andrew Lumsdaine. A component architecture for lam/mpi. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 379–387. Springer, 2003.
- [SMB<sup>+</sup>07] Laura J Scott, Karen L Mohlke, Lori L Bonnycastle, Cristen J Willer, Yun Li, William L Duren, Michael R Erdos, Heather M Stringham, Peter S Chines, Anne U Jackson, Ludmila Prokunina-Olsson, Chia-Jen Ding, Amy J Swift, Narisu Narisu, Tianle Hu, Randall Pruiam, Rui Xiao, Xiao-Yi Li, Karen N Conneely, Nancy L Riebow, Andrew G Sprau, Maurine Tong, Peggy P White, Kurt N Hetrick, Michael W Barnhart, Craig W Bark, Janet L Goldstein, Lee Watkins, Fang Xiang, Jouko Saramies, Thomas A Buchanan, Richard M Watanabe, Timo T Valle, Leena Kinnunen, Gonçalo R Abecasis, Elizabeth W Pugh, Kimberly F Doheny, Richard N Bergman, Jaakko Tuomilehto, Francis S Collins, and Michael Boehnke. A genome-wide association study of type 2 diabetes in Finns detects multiple susceptibility variants. *Science (New York, N.Y.)*, 316:1341–1345, 2007.
- [SMK<sup>+</sup>01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
- [Spe04] Charles Spearman. Spearman ’ s rank correlation coefficient. *Amer. J. Psychol.*, 15:72–101, 1904.
- [SRF87] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. 1987.
- [SRH14] Axel Schumacher, Tamas Rujan, and Jens Hoefkens. A collaborative approach to develop a multi-omics data analytics platform for translational research. *Applied & Translational Genomics*, 2014.

- [SS92] Graham R Serjeant and Beryl E Serjeant. *Sickle cell disease*, volume 3. Oxford university press New York, 1992.
- [SS11] The Sam and Format Specification. The SAM Format Specification (v1.4-r985). *Read*, pages 1–11, 2011.
- [SSC<sup>+</sup>08] Adam Silberstein, Adam Silberstein, Brian F. Cooper, Brian F. Cooper, Utkarsh Srivastava, Utkarsh Srivastava, Erik Vee, Erik Vee, Ramana Yerneni, Ramana Yerneni, Raghu Ramakrishnan, and Raghu Ramakrishnan. PNUTS: Yahoo!’s Hosted Data Serving Platform. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD ’08*, page 765, 2008.
- [SSDB95] Mark Schena, Dari Shalon, Ronald W Davis, and Patrick O Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, 1995.
- [SSH<sup>+</sup>01] Marco Salemi, Korbinian Strimmer, William W Hall, Margaret Duffy, Eric Delaporte, Souleymane Mboup, Martine Peeters, and Anne-Mieke Vandamme. Dating the common ancestor of sivcpz and hiv-1 group m and the origin of hiv-1 subtypes using a new method to uncover clock-like molecular evolution. *The FASEB Journal*, 15(2):276–278, 2001.
- [SSR11] Barbara E Stranger, Eli A Stahl, and Towfique Raj. Progress and promise of genome-wide association studies for human complex trait genetics. *Genetics*, 187:367–383, 2011.
- [SULS09] Marc Seeger and S Ultra-Large-Sites. Key-value stores: a practical overview. *Computer Science and Media, Stuttgart*, 2009.

- [SWK<sup>+</sup>01] S T Sherry, M H Ward, M Kholodov, J Baker, L Phan, E M Smigielski, and K Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29:308–311, 2001.
- [TB11] Bogdan George Tudorica and Cristian Bucur. A comparison between several nosql databases with comments and notes. In *Roedunet International Conference (RoEduNet), 2011 10th*, pages 1–5. IEEE, 2011.
- [TGAD<sup>+</sup>11] Sonia Tarazona, Fernando García-Alcalde, Joaquín Dopazo, Alberto Ferrer, and Ana Conesa. Differential expression in rna-seq: a matter of depth. *Genome research*, 21(12):2213–2223, 2011.
- [The05] The International HapMap Consortium. A haplotype map of the human genome, 2005.
- [The10] The International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. *Nature*, 467:52–58, 2010.
- [TPA<sup>+</sup>00] Svetlana Temnykh, William D Park, Nicola Ayres, Sam Cartinhour, N Hauck, Leonard Lipovich, Yong Gu Cho, Takashige Ishii, and Susan R McCOUCH. Mapping and genome organization of microsatellite sequences in rice (*oryza sativa* l.). *Theoretical and Applied Genetics*, 100(5):697–712, 2000.
- [TS93] MR Thomas and NS Scott. Microsatellite repeats in grapevine reveal dna polymorphisms when analysed as sequence-tagged sites (stss). *Theoretical and Applied Genetics*, 86(8):985–990, 1993.
- [TWM<sup>+</sup>12] Eleni Tsitsiou, Andrew E Williams, Sterghios A Moschos, Ketan Patel, Christos Rossios, Xiaoying Jiang, Oona-Delpuech Adams, Patricia Macedo, Richard Booton, David Gibeon, et al. Transcriptome analysis



- shows activation of circulating cd8+ t cells in patients with severe asthma. *Journal of Allergy and Clinical Immunology*, 129(1):95–103, 2012.
- [VAM<sup>+</sup>01] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.
- [VDW<sup>+</sup>08] Mandy Van Hoek, Abbas Dehghan, Jacqueline C M Witteman, Cornelia M. Van Duijn, André G. Uitterlinden, Ben A. Oostra, Albert Hofman, Eric J G Sijbrands, and A. Cecile J W Janssens. Predicting type 2 diabetes based on polymorphisms from genome-wide association studies A population-based study. *Diabetes*, 57:3122–3128, 2008.
- [VS11] Himanshu Vashishtha and Eleni Stroulia. *Enhancing query support in hbase via an extended coprocessors framework*. Springer, 2011.
- [WDH<sup>+</sup>02] James L Weber, Donna David, Jeremy Heil, Ying Fan, Chengfeng Zhao, and Gabor Marth. Human diallelic insertion/deletion polymorphisms. *The American Journal of Human Genetics*, 71(4):854–862, 2002.
- [WGS09] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009.
- [Whi12] Tom White. *Hadoop: The definitive guide*, volume 54. 2012.
- [Wil96] S Williams. Pearson’s correlation coefficient., 1996.
- [WPE<sup>+</sup>14] Shicai Wang, Ioannis Pandis, Ibrahim Emam, David Johnson, Florian Guittton, Axel Oehmichen, and Yike Guo. Dsimbench: A benchmark for microarray data using r. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 47–56. Springer, 2014.

- [WPJ<sup>+</sup>14] Shicai Wang, Ioannis Pandis, David Johnson, Ibrahim Emam, Florian Guitton, Axel Oehmichen, and Yike Guo. Optimising parallel r correlation matrix calculations on gene expression data using mapreduce. *BMC bioinformatics*, 15(1):351, 2014.
- [WPW<sup>+</sup>14] Shicai Wang, Ioannis Pandis, Chao Wu, Sijin He, David Johnson, Ibrahim Emam, Florian Guitton, and Yike Guo. High dimensional biological data retrieval optimization with NoSQL technology. *BMC genomics*, 15 Suppl 8(Suppl 8):S3, November 2014.
- [YMO<sup>+</sup>10] K Yamaguchi, M Mandai, T Oura, N Matsumura, J Hamanishi, T Baba, S Matsui, SK Murphy, and I Konishi. Identification of an ovarian clear cell carcinoma gene signature that reflects inherent disease biology and the carcinogenic processes. *Oncogene*, 29(12):1741–1752, 2010.
- [YTL<sup>+</sup>14] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. Druid: a real-time analytical data store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 157–168. ACM, 2014.
- [Yu02] Hao Yu. Rmpi: Parallel Statistical Computing in R. *R News*, 2:10–14, 2002.
- [ZC96] Fang L Zhang and Patrick J Casey. Protein prenylation: molecular mechanisms and functional consequences. *Annual review of biochemistry*, 65(1):241–269, 1996.
- [ZCF<sup>+</sup>10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

- [ZKJ<sup>+</sup>01] Ben Yanbin Zhao, John Kubiawicz, Anthony D Joseph, et al. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. 2001.
- [ZLN10] Weiwen Zhang, Feng Li, and Lei Nie. Integrating multiple 'omics' analysis for microbial biology: Application and methodologies, 2010.
- [ZLW<sup>+</sup>10] Yongqiang Zou, Jia Liu, Shicai Wang, Li Zha, and Zhiwei Xu. CCIndex : A Complemental Clustering Index on Distributed Ordered Tables for Multi-dimensional Range. In *the 9th IFIP International Conference on Network and Parallel Computing*, pages 247–261, 2010.
- [ZXW11] Yi-Hui Zhou, Kai Xia, and Fred A Wright. A powerful and flexible approach to the analysis of rna sequence count data. *Bioinformatics*, 27(19):2672–2678, 2011.