Bi-Dimensional Binning for Big Genomic Datasets

Simone Cattani
Politecnico di Milano, Milan, Italy
simone.cattani@polimi.it

Stefano Ceri
Politecnico di Milano, Milan, Italy
stefano.ceri@polimi.it

Abdulrahman Kaitoua
Politecnico di Milano, Milan, Italy
abdulrahman.kaitoua@polimi.it

Pietro Pinoli
Politecnico di Milano, Milan, Italy
pietro.pinoli@polimi.it

ABSTRACT

Binning the genome is used in order to parallelize big data operations upon regions. In this extended abstract, we comparatively evaluate the performance and scalability of Spark and SciDB implementations over datasets consisting of billions of genomic regions. In particular, we introduce an original method for binning the genome, i.e. partitioning it into sections of small sizes, and show that it outperforms conventional binning used by SciDB and closes the gap between SciDB and a Spark-based implementation. The concept of bi-dimensional binning is new and can be extended to other systems and technologies.

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

We are currently developing a new, holistic approach to genomic data modelling and querying. Our approach is based on GenoMetric Query Language (GMQL) [Masseroli et al. 2015], a high-level query language for genomics; the current implementations of GMQL[1], described in [Kaitoua et al. 2016], use Flink[2] and Spark[3] We recently implemented GMQL on a scientific data management system; we selected SciDB because it supports multi-dimensional data aggregation and because it includes an add-on specifically dedicated to tertiary data analysis for genomics [Paradigm4 2015a]; thus, it is an ideal alternative implementation framework for GMQL.

In [Cattani 2016] we benchmarked Spark and SciDB at work on genomic queries; the benchmark demonstrates the superiority of SciDB in computations which perform selections and aggregations, but also shows that Spark outperforms SciDB in *genomic map computations*, that perform genome-wise region comparisons (similar to join); in such cases, both the SciDB and Spark computations use the conventional *monodimensional binning*, a method for partitioning the genome into disjoint portions so as to enable parallelism. In this paper, we introduce a new method for binning, called *bidimensional binning*, that can be used also with cloud-based solutions but is best suited to the multidimensional characteristics of SciDB; we prove the correctness of bidimensional binning and show that bidimensional binning outperforms monodimensional binning in large size SciDB computations and reduces the gap in performance between SciDB and Spark.

2. GENOMIC ABSTRACTIONS FOR SCIDB

2.1 Data Model

We summarize the relevant features of Genomic Data Model (GDM), from [Kaitoua et al. 2016]. A *sample* s is a triple $\langle id, R, M \rangle$ where:

- id is the sample identifier; sample identifiers are unique within each dataset.
- R is the set of regions of the sample, built as pairs $\langle c, f \rangle$ of coordinates c and features f; coordinates are arrays of four fixed attributes *chr*, *left*, *right*, *strand*, features are arrays of typed attributes; we assume attribute names of features to be different. The *region schema* of s is the list of attribute names and types used for the identifier, the coordinates and the features.
- M is the set of metadata of the sample, built as attribute-value pairs $\langle a, v \rangle$.

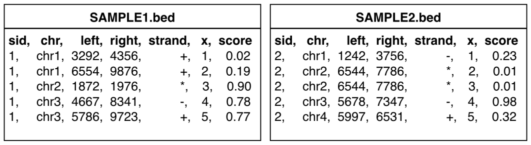
A *dataset* is a collection of samples with the same region schema. In this paper, we do not use metadata, so we do not further discuss how they are handled in SciDB. Regions of a dataset are stored into a single SciDB array; they are organized according to the relative sample id and genomic coordinate. We cast chromosome and strand values, natively represented by strings, to integers; regions data are mapped to a 6-dimensional array, where attribute fields are based on the specific feature schema.

```
DS_RD = < feature_schema >
    [sid, chr, left, right, strand, x]
```

The `x` dimension is an enumeration value, required because each GDM sample could have more than one region with the same coordinates and at least one different feature. Fig. 1

[1] www.bioinformatics.deib.polimi.it/GMQL/interfaces/
[2] www.apache.flink
[3] www.apache.spark

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235



SAMPLE1.bed

sid,	chr,	left,	right,	strand,	x,	score
1,	chr1,	3292,	4356,	+,	1,	0.02
1,	chr1,	6554,	9876,	+,	2,	0.19
1,	chr2,	1872,	1976,	*,	3,	0.90
1,	chr3,	4667,	8341,	-,	4,	0.78
1,	chr3,	5786,	9723,	+,	5,	0.77

SAMPLE2.bed

sid,	chr,	left,	right,	strand,	x,	score
2,	chr1,	1242,	3756,	-,	1,	0.23
2,	chr2,	6544,	7786,	*,	2,	0.01
2,	chr2,	6544,	7786,	*,	3,	0.01
2,	chr3,	5678,	7347,	-,	4,	0.98
2,	chr4,	5997,	6531,	+,	5,	0.32

Figure 1: Dataset with 2 samples in GDM format.

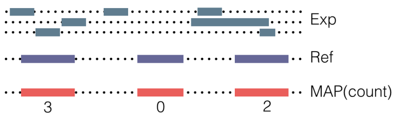


Figure 2: Mapping experiments to references in genomics.

shows a small dataset with two samples and a feature schema consisting of a single attribute.

Arrays are stored within fixed-size rectilinear **chunks** that partition the multidimensional space. Each chunk is then assigned to a computational node, using a hash function over the chunk's coordinates; the usage of region ends as coordinates allows their storage based on real region proximity, a fundamental property in order to speed up domain specific operations that use range intersection or range selection. According to [Paradigm4 2015b], the optimal size for a chunk should be between 5MB and 50MB. In our example, with a single attribute (and a size of about 8 Bytes), chunks with a million of regions have size of about 8-10 MB.

2.2 Genomic map

A region mapping operation applies to two datasets, called `Reference` and `Experiment` respectively. Although the problem formulation is generic, one can think to `Reference` region as known genome annotations (e.g. genes, exons, introns, enhancers) and to `Experiment` as regions produced by NGS processing (e.g. peaks of expressions or mutations). This operation performs the intersection of `Experiment` samples over the `Reference` and then computes an aggregate over such intersection (e.g., counts for each reference region how many experiment regions intersect with it). This behavior is explained in Figure 2, where we show a simple case consisting of one sample of `Reference` and one sample of `Experiment` with overlapping regions, where we count the number of experiment regions which intersect with each reference region (e.g., the third region of the `Reference` intersects with 2 regions of `Experiment` and therefore its count is 2).

A cloud-based implementation, e.g. in Spark, see [Kaitoua et al. 2016], consists of two main steps: (a) binning and (b) checking for intersection. In the binning phase, the genome is divided into bins and every region of both the `Reference` and the `Experiment` datasets is assigned to all the bins it overlaps. Then, the datasets are left-joined on the key: `(id,bin,chromosome)`. We use sort-merge join within the bins, by first sorting bins of the samples of both References and Experiments and then by merging bins by using a linear scan[4]. We output just the regions from a bin for which at least one of the starts of the two input regions is within the bin, thus avoiding to create duplicate regions in the result. This output generation condition generalizes a method presented in [Chawda et al. 2014]. Finally, adjacent regions on contiguous bins are aggregated (using a reduce phase), producing the final result. Monodimensional binning for parallelizing joins in map-reduce systems was introduced in [Chawda et al. 2014], and it was used in [Afrati et al. 2015] for assessing computational bounds; recent work include join methods for efficiently performing sort-based operators when regions do not overlap with two or more bins [Cafagna and Böhlen 2017] or for using features of modern CPU architectures [Piatov et al. 2016].

In the SciDB implementation [Cattani 2016] we initially adopted the above binning approach, but with an important difference. In SciDB it is not possible to dynamically split a region and distribute its replicas to an arbitrary number of adjacent bins, as we must apply identical operations to every cell in the array which stores the regions; thus, in order to apply a binning strategy, we must replicate all the regions an identical number of times. Such number is a function of the length `M` of the longest region in the `Reference` and `Experiment` datasets [Paradigm4 2015a] In general, for given `M` and bin size `S`, each region will span to at most `R` bins, with: $\mathtt{R} = \lceil \mathtt{M/S} \rceil + 1$. This is a limitation w.r.t. Spark, which can manage variable region replication; region replication in Spark occurs only when the region spans across two or more contiguous bins.

2.3 Benchmark

We performed our experiments on the Amazon Web Services (AWS) cloud, using a configuration with r3.4xlarge machine, 16 cores, 122 GB of RAM and 320GB of SDD. Testing was performed on datasets with an increasing number of samples (up to 2K) and regions; see Table 1. Execution times of region mapping in Spark and SciDB are reported in Table 2.

We note that Spark outperforms SciDB, whose performance rises to about 1.5 hrs when comparing .5 million regions of the reference with 101 million regions of 2000 experiments. For this reason, and given the limitations of SciDB binning algorithms discussed in Section 2.2, we decided to focus on a new method for genome binning, that better adapts to the computational model of SciDB, discussed in the next section.

Dataset	Size (MByte)	Regions (Million)	Samples
REF	2.3	0.506	1
DS_1	38	1.012	20
DS_2	375	10.120	200
DS_3	3832	101.2	2000

Table 1: Features of the datasets used in the benchmark.

Test	DS_1	DS_2	DS_3
Spark	0.12	0.57	3.82
SciDB	0.28	3.29	95.33

Table 2: Execution times (in minutes) for the genomic map operation.

[4] https://en.wikipedia.org/wiki/Sweep_line_algorithm

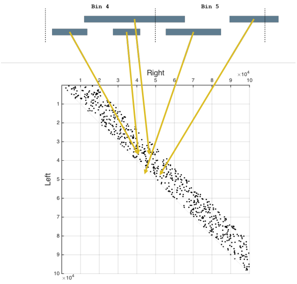


Figure 3: Region assignment to bins with bidimensional binning

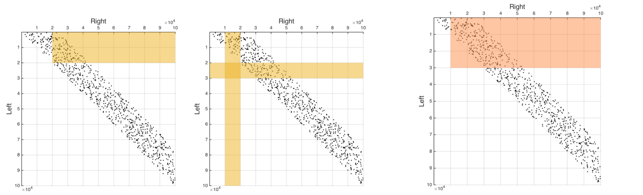


Figure 4: Experiment regions that certainly intersect with regions in bin (2,3); then experiment regions that can intersect with regions in bin (2,3); and then their composition as a target space for the regions of bin (2,3).

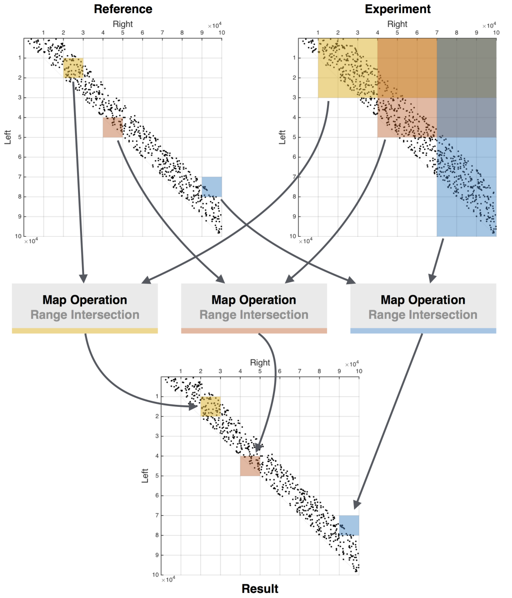


Figure 5: Bidimensional binning strategy

3. BIDIMENSIONAL BINNING

In this approach, each region `R` is assigned to a bin defined by a pair of identifiers:

$$\texttt{bin(R)} = \left(\left\lfloor \frac{R_{start}}{\texttt{bin\_size}} \right\rfloor , \left\lfloor \frac{R_{stop}}{\texttt{bin\_size}} \right\rfloor \right)$$

A region is assigned to the (n, m) bin when it starts in the *n-th* bin and ends in the *m-th* bin (see Fig. 3); note that the genome is partitioned into a bidimensional grid rather than a monodimensional vector, and that every region is mapped to a point in such grid; since each region is constrained to have `start` $<$ `stop`, a region can be assigned only to a cell either in the primary diagonal or above the diagonal of such space; in most genomic applications regions are short, hence the majority of points clusters either in the diagonal or in the cells immediately above the diagonal.

Consider now the mapping between a `Reference` and an `Experiment` dataset. For each bin in the reference we can divide the experiment regions into three groups: (a) regions that for sure intersect all the reference region in the bin, (b) regions that can potentially intersect them, and (c) regions that do not intersect them. Consider for example the bin $(2, 3)$ of the reference, i.e., regions that start in bin 2 and end in bin 3. Fig. 4 shows the regions of the experiment that certainly intersect with them, then the regions of the experiment that could possibly intersect with them; their composition (taking into account that no region falls below the diagonal) generates a rectangular *target space* for the bin $(2, 3)$ of the reference, also shown in Fig. 4. The following theorem generalises this example.

Theorem: Let $R = (l_R, r_R)$ be a box in the reference. The search space window that defines the subset of the experiment regions that can intersect with R is composed by all the boxes $E = (l_E, r_E)$ that verify the following condition:

$$(\ l_E \leq r_R\ ) \wedge (\ r_E \geq l_R\ ) \tag{1}$$

Proof. To prove the theorem is sufficient to show that when the above condition (1) is false then no intersection can occur; we enumerate all such cases:

- $(\ l_E > r_R\ ) \wedge (\ r_E \geq l_R\ )$. In this case, all the experiment regions start after the end of the reference, so their intersection is empty.
- $(\ l_E \leq r_R\ ) \wedge (\ r_E > l_R\ )$. In this case, all the experiment regions end before the start of the reference, so their intersection is empty.
- $(\ l_E > r_R\ ) \wedge (\ r_E < l_R\ )$. This case is impossible because it implies either $l_E > r_E$ or $l_R > r_R$, which is excluded by definition.

□

The overall bidimensional binning strategy for region mapping is illustrated in Fig. 5. In this approach, several independent queries are executed by a query controller written in Scala, one for each non-empty bin of the reference (the figure shows theee such queries). For a given bin of the reference, an AFL query computes range intersections with all the regions of the experiment which belong to the bin's target space; the result is an aggregate value, associated to the regions of the reference bin.

Implementing bi-dimensional binning in SciDB is simpler than conventional binning, as the SciDB operator `between` supports region filtering according to the condition expressed in Theorem 1, and references are joined with experiments according to the scheme of Fig. 5, which applies in parallel to all the non-empty reference regions above the diagonal. After the cross-join, result is redimensioned as a table with two dimensions, x-ref and chr, having in its cells values for all the experiments overlapping with each x-ref region, and aggregate functions over the experiments can be computed.

4. EVALUATION OF BINNING STRATEGIES

As first evaluation of bidimensional binning, we consider again query Q6 of Section 4.D; execution times are reported in Table 3; note that, when executed over the dataset DS_4, bidimensional binning improves of about a factor 3 over monodimensional binning, covering part of the difference in performance between SciDB and Spark.

Test	DS_1	DS_2	DS_3
Spark *Q6*	0.12	0.57	3.82
SciDB_D2 *Q6*	0.43	2.10	28.49
SciDB_D1 *Q6*	0.28	3.29	95.33

Table 3: Execution times of the mapping operation.

In order to better evaluate bidimensional binning, we then considered real datasets. For the references, we considered two different types of regions:

- **Genes** are heterogeneous regions, as their maximum length is 24187702, their minimum length is 19, their average length is 60680, and their median length is 20102. This length variability could negatively affect monodimensional binning.

- **Promoters** are small homogeneous regions, each of size 2999, artificially built around a specific genomic position, the *transcription start site*. This lenght regularity could instead favor monodimensional binning.

The experiments datasets are collected from ENCODE Narrow Peaks (NP) with different sizes, as shown in 4.

Dataset	Size (MB)	Regions (× K)	Samples
GENES	0.7	23.033	1
PROMOTERS	2.3	49.052	1
NP_1	17	363.537	2
NP_2	41	938.753	4
NP_3	57	1.264.764	8
NP_4	108	2.230.698	16
NP_5	155	3.227.907	32

Table 4: Features of the real datasets

Performance comparisons are shown in Fig. 6. Note that in all cases Spark outperforms SciDB, but the difference between Spark and SciDB with bidimensional binning is much reduced, and that Spark and SciDB curves scale in a similar way. Note as well that bidimensional bidding outperforms monodimensional binning in the three largest experiments; the two curves optimize monodimensional binning by setting region replication in SciDB to 8 for genes and to 2 for promoters, the best replication factors, obtained after several experiments (with suitable tuning, the performance of mono-dimensional binning have been improved by adapting to the lengths of reference regions). In our future work we plan to investigate the use of bidimensional binning with Spark.

Acknowledgment

This work is supported by the ERC Advanced Grant *GeCo (Data-Driven Genomic Computing)*.

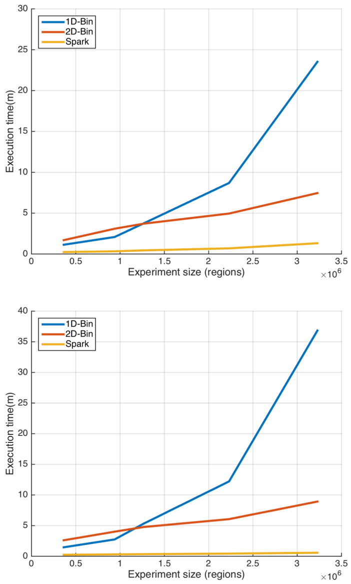


Figure 6: Performance comparison using genes and promoters as reference

5. REFERENCES

[Afrati et al. 2015] Foto N Afrati, Shlomi Dolev, Shantanu Sharma, and Jeffrey D Ullman. 2015. Bounds for Overlapping Interval Join on MapReduce.. In *EDBT/ICDT Workshops*. 3–6.

[Cafagna and Böhlen 2017] Francesco Cafagna and Michael H Böhlen. 2017. Disjoint interval partitioning. *The VLDB Journal* (2017), 1–20.

[Cattani 2016] Simone Cattani. 2016. Genomic computing with SciDB, a data management system for scientific applications. (2016).

[Chawda et al. 2014] Bhupesh Chawda, Himanshu Gupta, Sumit Negi, Tanveer A Faruquie, L Venkata Subramaniam, and Mukesh K Mohania. 2014. Processing Interval Joins On Map-Reduce.. In *EDBT*. 463–474.

[Kaitoua et al. 2016] Abdulrahman Kaitoua, Pietro Pinoli, Michele Bertoni, and Stefano Ceri. 2016. Framework for Supporting Genomic Operations. *IEEE Trans. Comput.* (2016).

[Masseroli et al. 2015] Marco Masseroli, Pietro Pinoli, Francesco Venco, Abdulrahman Kaitoua, Vahid Jalili, Fernando Palluzzi, Heiko Muller, and Stefano Ceri. 2015. GenoMetric Query Language: a novel approach to large-scale genomic data management. *Bioinformatics* 31, 12 (2015), 1881–1888.

[Paradigm4 2015a] Paradigm4. 2015a. Accelerating bioinformatics research with new software for big data to knowledge (BD2K).

[Paradigm4 2015b] Paradigm4. 2015b. SciDB MAC Storage Explained.

[Piatov et al. 2016] Danila Piatov, Sven Helmer, and Anton Dignös. 2016. An interval join optimized for modern hardware. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 1098–1109.