A Context-based Approach for Partitioning Big Data

Sara Migliorini
Dept. of Computer Science, University of Verona
sara.migliorini@univr.it

Alberto Belussi
Dept. of Computer Science, University of Verona
alberto.belussi@univr.it

Elisa Quintarelli
Dept. of Computer Science, University of Verona
elisa.quintarelli@univr.it

Damiano Carra
Dept. of Computer Science, University of Verona
damiano.carra@univr.it

ABSTRACT

In recent years, the amount of available data keeps growing at fast rate, and it is therefore crucial to be able to process them in an efficient way. The level of parallelism in tools such as Hadoop or Spark is determined, among other things, by the partitioning applied to the dataset. A common method is to split the data into chunks considering the number of bytes. While this approach may work well for text-based batch processing, there are a number of cases where the dataset contains structured information, such as the time or the spatial coordinates, and one may be interested in exploiting such a structure to improve the partitioning. This could have an impact on the processing time and increase the overall resource usage efficiency.

This paper explores an approach based on the notion of context, such as temporal or spatial information, for partitioning the data. We design a context-based multi-dimensional partitioning technique that divides an $n-$dimensional space into splits by considering the distribution of the each contextual dimension in the dataset. We tested our approach on a dataset from a touristic scenario, and our experiments show that we are able to improve the efficiency of the resource usage.

KEYWORDS

Big Data, Partitioning, Contextual dimensions

1 INTRODUCTION

A dataset analyzed using parallel data processing systems, such as Hadoop or Spark, is usually divided into chunks, or *splits*. The basic partitioning approach uses the amount of bytes (e.g., 64 or 128 MB) as splitting technique, without considering the content of the data. In case of batch processing, where the dataset is always analyzed entirely, this solution is reasonable. Nevertheless, if the dataset is analyzed using selective queries based on some attributes of the data, like time intervals or spatial regions [6, 11], such an approach may not be efficient, since the partitioning does not exploit the correlations in the data.

Consider for instance a dataset that collects the visits of tourists at different Points of Interests (PoIs). The tourists have a city pass which they swipe at the entrance of a PoI. Each swipe contains the identifier of the city pass, the name and location (coordinates) of the visited PoI, together with an entrance timestamp. One may analyze such a dataset considering the timestamp (How many tourists have been there in a specific day?), or the space (How many times has a specific PoI been visited?), or PoI type (Are modern-art museums preferred to science museums?). One may also combine different dimensions (How many tourists visited a specific PoI in a specific hour?).

As a general approach, we consider *context-aware* partitioning techniques. Assuming that a dataset can be analyzed w.r.t. several dimensions, the idea is to group in the same split records that are context-related [7]. For instance, if the context is defined by the space and time dimensions, a context-aware partitioning will include in the same split records that are nearby to each other from both a spatial and temporal point of view.

Previous works proposed partitioning approaches mainly based on spatial [9, 12] and spatio-temporal [1, 2] characteristics. In case of spatial partitioning, one may partition based on space (grid and Quad-tree), based on data (STR, STR+, K-d tree), or based on space filling curves (Z-curve, Hilbert curve) [8]. The selection of the partitioning technique is usually left to the user, and only few works automatically select the best partitioning technique based on the dataset distribution [5].

When two or more dimensions need to be combined, there are two possible approaches. The first one considers each dimension independently and builds a *multi-level partitioning*. This approach produces a list of n grids (one for each level) that are used for performing the partitioning, and it imposes an order between them. The chosen order can have a great impact on the nature and balancing of the resulting splits. For instance, ST-Hadoop [1] firstly divide the dataset based on temporal granularity, and then splits each portion based on spatial proximity. A query focused on spatial properties (e.g., Has PoI x been visited more than PoI y?) requires the analysis of all, temporally organized, splits.

The second approach considers all the dimensions together and builds a *multi-dimensional partition*, i.e., a n-dimensional grid. An example is HadoopTrajectory [2], in which partitions are 3D cubes where the three dimensions are space (planar coordinates) and time. Given a query focused on one dimension, this approach allows the exact selection of the splits that could be useful in answering the query. The challenge imposed by the multi-dimensional partitioning is to find the best size of the grid cells in each dimension, so that the amount of data in each cell is balanced. This can be a non trivial task, especially in the general case where data are not uniformly distributed [3, 5].

In this paper, we consider a context-based multi-dimensional partitioning approach, which takes as input a dataset D and the set of n contextual dimensions relevant to analyse D. We design a solution that automatically produces the most appropriate division of the $n-$dimensional space, considering the distribution of each contextual dimension inside D. The proposed technique could be adopted in case of recurrent queries, to drive the partitioning of the dataset that is stored permanently (e.g., HDFS), or it can be used in a dynamic scenario, where the dataset is kept in-memory (e.g., Spark), and it can be repartitioned based on the current set of queries.

We evaluate our solution on a real-world dataset containing the swipes of a city pass – with the characteristics previously described. The results show that we are able exploit the partitioning to efficiently process a set of representative queries.

2 PROBLEM FORMULATION

This section formalizes the context-aware partitioning problem.

Definition 2.1 (Dataset). A *dataset schema* $S = \langle a_1, \ldots, a_m \rangle$ is a list of attributes, each one belonging to a particular domain, denoted as $\Delta(a_i)$. A *dataset* $D = \{r_1, \ldots, r_n\}$ over a schema S is a collections of records $r_i = \langle v_1, \ldots, v_m \rangle$, where $\forall i \in \{1, \ldots, m\}\ v_i \in \Delta(a_i)$.

Definition 2.2 (Context). Given a dataset D over a schema $S = \langle a_1, \ldots, a_m \rangle$, the context is a subset of the attributes in S:

$$C = \{c_1, \ldots, c_k\} \subseteq \{a_1, \ldots, a_m\} \tag{1}$$

Definition 2.3 (Partitioning). Given a dataset $D = \{r_1, \ldots, r_n\}$, a partitioning P is a collection of subsets of D:

$$P = \{p_1, \ldots, p_h\} \text{ such that } \forall p_i \in P\,(p_i \subseteq D) \text{ and } D = \cup_i p_i \tag{2}$$

Definition 2.4 (Balanced partitioning). A partitioning $P = \{p_1, \ldots, p_h\}$ for a dataset D is said to be balanced if and only if:

$$\forall p_i, p_j \in P:\ abs(|p_i| - |p_j|) \leq \varepsilon \tag{3}$$

where $|p_i|$ denotes the cardinality of the partition p_i.

Definition 2.5 (Context-aware partitioning). A partitioning P_C for a dataset D is the minimal one for the context C and a context-based query q if and only if:

(1) it is a balanced partitioning, i.e. it is able to produce balanced partitions,
(2) it minimizes the number of splits to consider for answering the query q. Notice that a partition might contain also more than one split.

3 CONTEXT-BASED PARTITIONING

In order to partition a dataset D considering a context C with several dimensions of analysis, different approaches can be applied. For instance, in the multi-dimensional partitioning the subdivision of the elements in D is performed by defining an n-dimensional grid where $n = |C|$. Conversely, a multi-level partitioning could also be applied by using n grids, each one representing a level and corresponding to a dimension in C.

Notice that given a dataset D, a context C and a query q, it is possible to have multiple minimal partitionings. Indeed, the quality of the resulting set of partitions can highly depend on the distribution in D of the values of the dimensions belonging to C and those used in q. When our partitioning technique is not applied on-line before executing each query q, the minimality of a partitioning can be evaluated by considering the average number of splits used for a reference set of context-based queries.

Given a context C for a dataset D, the identification of the most appropriate partitioning requires an easy and efficient way for evaluating the skewness in D of each context dimension. Based on this evaluation, the right shape of each n-dimensional cell inside the n-dimensional grid can be determined.

The aim of this paper is to propose a partitioning technique able to capture the distribution of the dataset w.r.t. each context dimension and based on this to build the right set of partitions even for skewed datasets. For this purpose, we extend the idea originally proposed in [5] for the spatial domain to the management of a generic number n of context dimensions. For this reason, we present below the definition of the box-counting function $BC_r^q(D, a)$ for a given dataset D and a context dimension a, that is the fundamental notion for the skewness evaluation.

Definition 3.1 (Box-counting function for a dimension a). Given a dataset D, containing an attribute a belonging to a domain $\Delta(a)$, and a scale r representing the cell size of a mono-dimensional grid covering the range of values of $\Delta(a)$ appearing in D, the box-counting function $BC_r^q(D, a)$ is defined as:

$$BC_r^q(D, a) = \sum_i \delta_i(D, a)^q \quad \text{with } q \neq 1 \tag{4}$$

where $\delta_i(D, a)$ is the number of records in D whose value for a is contained in the cell i. The case $q = 1$ is excluded, since it does not depend on r and it equals the number of records in D. □

Intuitively, given a grid with cells of side r, the box-counting function with $q = 0$ counts the number of cells that contains at least one record of D. When q is greater than 1, the box-counting becomes the sum of the number of records that a cell contains, raised to q. This function can be used to detect the skewness of a dataset by computing it for $q = 0$ and $q = 2$, while varying the value of r. More specifically, the level of skewness of a dataset depends on how this value changes while increasing r.

Definition 3.2 (Box-counting plot). Given a dataset D, containing an attribute a belonging to a domain $\Delta(a)$, the box-counting plot is the plot of $BC_r^q(D, a)$ versus r in logarithmic scale.

On datasets representing fractals, since it can be derived from theory, and on real datasets, as shown in [5, 10], the following observation can be considered valid.

Observation 1. *For finite datasets representing fractals and real datasets the box counting plot reveals a trend of the box counting function that, in a large interval of scale values r, behaves as power law:*

$$BC_r^q(D, a) = \alpha \cdot r^{E_q} \tag{5}$$

where α is a constant of proportionality and E_q is a fixed exponent that characterizes the power law.

The power law exponent E_q for a given dataset D and an attribute a can be computed by starting from the box-counting plot, since it becomes the slope of the strait line that approximates $BC_r^q(D, a)$ in a range of scale (r_1, r_2), thus it can be computed by a linear regression procedure.

We can observe that E_0 and E_2 could be chosen as reference descriptors about the distribution of the values of the attribute a in the dataset D. Indeed, E_0 can be an indicator of the cases where the dataset leaves empty some areas of the range of values of a covered by D, while E_2 can also be affected by the concentration of the values in some areas with respect to other ones, i.e. the situations where there are no empty areas, but different data concentrations in different areas.

In order to optimize the computation of E_0 and E_2 for a big dataset D_{big} the following approach can be applied. First, we consider a sample of D_{big} (usually 10% of the records) for the computation of the n-dimensional histogram: it is composed of a n-dimensional cube counting the number of records falling in each cell (only the non-empty cells are represented). Second, the projection of the n-dimensional cube on each dimension produces n one-dimensional histograms to be used for the computation of E_0 and E_2 for each dimension. Third, considering the heuristics presented in [5], accordingly to E_0 and E_2 the suitable partitioning technique for each dimension is chosen (possible techniques are: regular grid, space-based partitioning, record-based partitioning). Finally, D_{big} is scanned and partitioned using the n-cube produced by intersecting the list of splitting planes obtained by applying the chosen techniques. Notice that for each dimension a different technique might have been chosen.

4 CASE STUDY EVALUATION

The proposed technique has been applied to a real-world dataset containing the swipes of a city pass, called *VeronaCard*, which is provided by the tourist office of Verona, a municipality in Northern Italy. The dataset contains about 1,200,000 records concerning 4 years. Each record reports beside to the identifier of the city pass and the name of the visited PoI: the location (coordinates) of the PoI, the entrance timestamp and the age of the tourist holding the card.

Fig. 1 shows the spatial distribution of the records: the size of the circle surrounding the PoI name represents the number of records regarding that location: bigger cycles represent PoIs with the higher number of visits. As you can notice, the records are not uniformely distributed w.r.t. space, since there are some PoIs, such as *Arena* and *Casa di Giulietta*, which have much more visits w.r.t. others, like *San Fermo*.

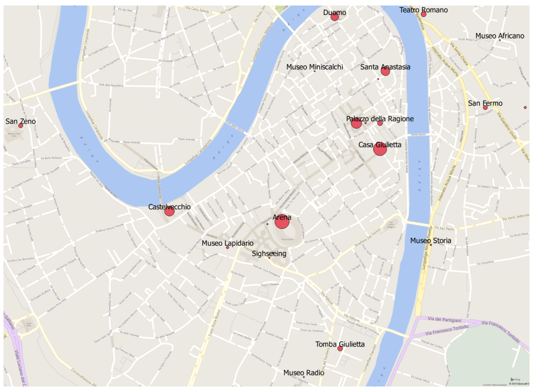


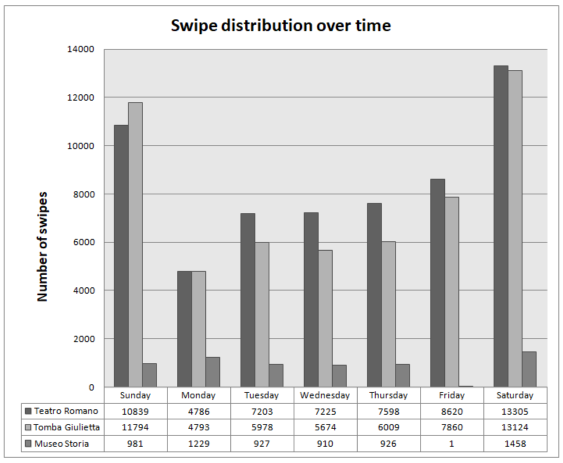
Figure 1: Spatial distribution of the swipes: bigger circlers represent an higher number of visits in that PoI (records).

A sample about the distribution of the records w.r.t. the time is represented in Fig. 2 by means of histograms. In particular, we show the distribution of three PoIs aggregated by the day of the week. Even in this case the distribution is not uniform, since there are some days in which a PoI is mostly visited than others (es. weekend days vs week days). Moreover, some PoI can have a closing day in which there are no visits at all.

Finally, the age distribution is reported in Fig. 3 by means of a Pie chart. Even in this case the distribution is not uniform: some ages more frequent than others, reflecting the fact that there are PoIs more suitable for some kinds of tourists than others.

These different distributions are recognized also by the exponents E_0 and E_2 of their box counting plot, as reported in Tab. 1 together with the chosen partitioning technique.

Tab. 2 illustrates some statistics about the partitions produced by the four considered partitioning techniques: *Rand* is the default partitioning technique traditionally supported by a MapReduce environment, it simply subdivides the dataset into parts with homogeneous size in bytes. MD_{grid} is a multi-dimensional uniform grid partitioning technique which essentially subdivides the dataset by using uniform d-dimensional cells for data aggregation, while ML_{grid} is a multi-level uniform grid partitioning



	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
■ Teatro Romano	10839	4786	7203	7225	7598	8620	13305
▨ Tomba Giulietta	11794	4793	5978	5674	6009	7860	13124
■ Museo Storia	981	1229	927	910	926	1	1458

Figure 2: Number of swipes for 3 PoIs by day of the week.

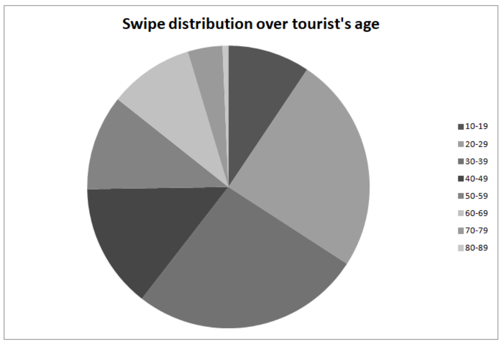


Figure 3: Number of swipes for each age value.

Table 1: Exponents E_0 and E_2 and technique choice.

Context attribute	$-E_0$	E_2	Index
time	1.041	0.963	Regular Grid
x	0.041	0.001	Space-based partitioning
y	0.054	0.154	Space-based partitioning
age	0.506	0.474	Record-based partitioning

which considers only a dimension at each phase during the uniform subdivision. Finally, *CBP* is the context-based partitioning technique proposed in this paper which uses the computation of the box counting plot and the corresponding exponents E_0 and E_2, in order to produce the most appropriate d-dimensional grid in order to accomodate also non uniform data.

Table 2: Experimental results: RAND is the random partitioning, %RSD is the relative standard deviation with respect to the dimension analysis.

Index	#parts	#splits	%RSD	%RSD	%RSD	%RSD	%RSD
			#size	time	x	y	age
Rand	69	69	1%	57%	33%	35%	58%
MD_{grid}	64	109	64%	25%	4%	17%	24%
ML_{grid}	73	117	71%	26%	4%	14%	23%
CBP	88	88	26%	20%	2%	3%	19%

The first consideration we can done is that while *Rand* is able to produce the minimum number of splits with respect to the dataset dimension and the split size (#parts = #splits), the other techniques can produce a greater number of splits which is equal to: $\lceil \sqrt[d]{ds/sp} \rceil$ where d is the number of dimensions, ds is the dataset size in bytes and sp is the split size in bytes. Notice that since the dataset is not uniformly distributed: (1) some cells could be empty, so they are not produced (i.e., MD_{grid} has 64 partitions and ML_{grid} has 73 partitions). (2) Some cells could be overpopulated, so they have to be split in order to comply with the split size prescription, i.e., at the and MD_{grid} has 109 splits, while ML_{grid} has 117 splits. Conversely, *Rand* and *CBP* has a number of splits equal to the number computed of partitions. *CBP* produces more partitions than *Rand* due to the additional subdivision of the *n*-dimensional space in case of clustered data.

The non uniformity of the dataset distribution has a direct effect also on the split size variability. In order to evaluate the variability of a given feature, we use the %RSD, which is the relative standard deviation, namely it is a statistical measurement describing the spread of data with respect to the mean and the result is expressed as a percentage. Clearly, the *Rand* technique is able to produce very balanced splits, since the partitioning is guided only by this parameter. Conversely, MD_{grid} and ML_{grid} produce very unbalanced splits due to the data skeweness. With *CBP*, we obtain quite good results in terms of balancing.

The second aspect to consider is the variability of each dimension values inside the splits. Columns 5-8 in Tab. 2 reports the average %RSD for each considered dimension. As you can notice, all the last three techniques generally improve the performances of the *Rand* technique. In particular, as regards to the temporal dimension, since it is quite uniformly distributed and also it is the first considered dimension by the ML_{grid}, its average spread is the same for MD_{grid} and ML_{grid}, while for the other dimensions the spread is less for MD_{grid} since it considers all dimensions at the same level for producing the partitioning. *CBP* produces splits with less variability in each dimension w.r.t. all the other techniques.

Given the partitions induced by the four considered partitioning techniques, we have performed some representative range queries in order to evaluate their performances. The performances are evaluated as the number of splits that have to be processed in order to produce the desired result, namely in the filtering capabilities induced by each partitioning technique.

We consider 4 queries, the first one has a condition on all the context dimensions, while the other ones contain conditions on less dimensions. The results are reported in Tab. 3.

Q_1: *find all visits performed around the Arena during spring 2015 by young tourists.* The spatial location is defined by a buffer around the Arena, while the period *spring 2015* is defined by a temporal interval, and the *young tourists* are identified by an age range. Since the condition regards all the four dimensions, the performance of MD_{grid} and ML_{grid} are quite similar.

Q_2: *find all visits performed by teenager in 2016 everywhere in Verona.* In this case the spatial dimensions are not considered in the filter, while a pruning is performed on the temporal dimension. Since this represents the first level for ML_{grid}, it sightly outperforms MD_{grid}.

Q_3: *find all visits around Arena performed by senior tourists.* In this case the temporal dimension is not considered, while the spatial and age dimensions are considered. Differently to the previous case, the advantage of ML_{grid} is loss, since no filtering can be performed at the first level.

Q_4: *find all the visits performed by adult tourists.* As you can notice here the filter is applied only on the 4th dimension, so

Table 3: Experimental results. Numbers represents #splits.

Query	*Rand*	MD_{grid}	ML_{grid}	*CBP*
Q1	69	3	3	2
Q2	69	32	30	14
Q3	69	6	6	5
Q4	69	85	94	80

ML_{grid} performs worst than MD_{grid}, because it has to scan all the levels before applying a filter.

5 CONCLUSION

In MapReduce frameworks the partitioning of a dataset into independent splits is a critical operation, since the degree of parallelism and the overall performances directly depend from the initial partitioning technique. This is particularly true in case of context-based applications, where data present correlations and consequently data could be aggregated and filtered in order to reduce the amount of work to be done during the analysis. Moreover, beside the need for a context-based partitioning technique, in order to produce balanced splits, it is necessary to consider the distribution of the dataset w.r.t. the analysis dimensions. This paper proposes a context-based partitioning technique which takes care of the dimension distributions to produce the best partitioning for the dataset at hand. We also apply it to a real-world dataset and compare its performances w.r.t. existing partitioning techniques for highlighting its differences and benefits. The obtained preliminary results confirm the goodness of the approach and encourage further research in this direction, for instance as regards to the managament of multi-accuracy data [4].

ACKNOWLEDGMENTS

This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM). and by "Progetto di Eccellenza" of the Computer Science Dept., Univ. of Verona, Italy.

REFERENCES

[1] L. Alarabi, M. F. Mokbel, and M. Musleh. 2018. ST-Hadoop: a MapReduce framework for spatio-temporal data. *GeoInformatica* 22, 4 (2018), 785–813.

[2] M. Bakli, M. Sakr, and Taysir H. A. S. 2019. HadoopTrajectory: a Hadoop spatiotemporal data processing extension. *Journal of Geographical Systems* 21, 2 (2019), 211–235.

[3] A. Belussi, D. Carra, S. Migliorini, M. Negri, and G. Pelagatti. 2018. What Makes Spatial Data Big? A Discussion on How to Partition Spatial Data. In *10th Int. Conf. on Geographic Information Science (GIScience 2018).* 2:1–2:15.

[4] A. Belussi and S. Migliorini. 2012. A Framework for Integrating Multi-Accuracy Spatial Data in Geographical Applications. *Geoinformatica* 16, 3 (2012), 523–561.

[5] A. Belussi, S. Migliorini, and A. Eldawy. 2018. Detecting Skewness of Big Spatial Data in SpatialHadoop. In *Proceedings of the 26th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems.* 432–435.

[6] A. Belussi, S. Migliorini, M. Negri, and G. Pelagatti. 2015. Validation of Spatial Integrity Constraints in City Models. In *Proc. of the 4th ACM SIGSPATIAL Int. Workshop on Mobile Geographic Information Systems.* 70–79.

[7] C. Bolchini, E. Quintarelli, and L. Tanca. 2013. CARVE: Context-aware automatic view definition over relational databases. *Inf. Syst.* 38, 1 (2013), 45–67.

[8] A. Eldawy, L. Alarabi, and M. F. Mokbel. 2015. Spatial Partitioning Techniques in SpatialHadoop. *Proc. VLDB Endow.* 8, 12 (2015), 1602–1605.

[9] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *2015 IEEE 31st Int. Conf. on Data Engineering.* 1352–1363.

[10] C. Faloutsos, B. Seeger, A. Traina, and C. Traina, Jr. 2000. Spatial Join Selectivity Using Power Laws. *SIGMOD Rec.* 29, 2 (2000), 177–188.

[11] S. Migliorini, A. Belussi, M. Negri, and G. Pelagatti. 2016. Towards Massive Spatial Data Validation with SpatialHadoop. In *Proc. of the 5th ACM SIGSPATIAL Int. Workshop on Analytics for Big Geospatial Data.* 18–27.

[12] J. Yu, Z. Zhang, and M. Sarwat. 2019. Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond. *Geoinformatica* 23, 1 (2019), 37–78.