LUCAS HENRIQUE MOREIRA SILVA

Advisor: Joubert de Castro Lima

COMPUTING DATA CUBES OVER GPU CLUSTERS

Ouro Preto

June of 2018

Federal University of Ouro Preto
Institute of Exact Sciences
Graduate Program in Computer Science

COMPUTING DATA CUBES OVER GPU CLUSTERS

Monograph presented to the Graduate Program in Computer Science of the Federal University of Ouro Preto in partial fulfillment of the requirements for the degree of in Computer Science.

LUCAS HENRIQUE MOREIRA SILVA

Ouro Preto
June 2018

FEDERAL UNIVERSITY OF OURO PRETO

CERTIFICATE OF APPROVAL

Computing Data Cubes Over GPU Clusters

LUCAS HENRIQUE MOREIRA SILVA

Monograph defended and approved by the examination board composed by:

Ouro Preto, June of 2018

Resumo

O cubo de dados é um operador relacional fundamental para sistemas de suporte a tomada de decisão, dessa forma de extrema importância para análise de Big Data. O problema apresentando nesse trabalho é: como reduzir os tempos de resposta de consultas multidimensionais complexas? Tal problema se tona ainda mais agravado se atualizações recorrentes nos dados de entrada acontecem e se existe um grade volume de dados de alta dimensionalidade a ser analisado. A hipótese deste trabalho é que uso de clusters de dispositivos CPU-GPU irá acelerar consultas em cubos de dados holísticos de alta dimensão que são constantemente atualizados. Esse cenário é muito próximo dos requisitos de Big Data, portanto muito próximo de cenários modernos de banco de dados. A solução alternativa proposta neste trabalho, chamada de JCL-GPU-Cubing, particiona a base de dados em múltiplas representações de cubos parciais sem introduzir redundância de dados e pontos de sincronização. Tais cubos parciais são usados para executar consultas em CPU ou CPU-GPU de maneira eficiente. As avaliações experimentais preliminares demonstraram que a versão baseada em clusters de CPU escala bem quando ambos os dados de entrada e o tamanho do cluster aumentam. Este trabalho representa a primeira parte de um trabalho mais complexo com entrega em dezembro de 2018. Dessa forma, várias melhorias serão feitas até o final deste ano, incluindo novos experimentos comparativos, detalhamento dos algorítimos , exemplos com figuras e discussões em profundidade sobre os pontos fortes e desvantagens do JCL-GPU-Cubing

Palavras-chave: GPU, OLAP, data cube, distributed computing, parallel computing, big data

Abstract

The data cube is a fundamental relational operator for decision support systems, this way very important for big data analytics. The problem stated in this work is: how to reduce complex multidimensional queries response times? The problem is aggravated if recurrent updates in the input data occurs and if there is a huge volume of high dimensional data to be analyzed. The hypothesis of this work is that clusters of CPU-GPU devices can speedup queries from huge high dimensional holistic data cubes that are updated constantly. This scenario is very close with the big data requirements, thus very close to modern database scenarios. The alternative solution presented in this work, named JCL-GPU-Cubing, partitions the base relation into multiple partial cube representations without introducing data redundancies and synchronization points. These partial data cubes are used to perform queries in CPU or in CPU-GPU efficiently. The initial experimental evaluations demonstrated that the CPU cluster based version scaled well when both input data and number of devices increased. This work represents the first part of a more complex work with deadline at December, 2018. This way, many improvements will be made until the end of this year, including new comparative experiments, the algorithms in detail, examples with figures and deep discussions about JCL-GPU-Cubing strengths and drawbacks.

keywords: GPU, OLAP, data cube, distributed computing, parallel computing, big data

I dedicate this work to my Father, Nereu and Mother, Marilene, who taught me that one should not brag about his accomplishments, but let them shine through one's hard work

Acknowledgments

None of the accomplishments of this work would be possible without my advisor, Dr. Joubert C. Lima. I must thank him for the opportunity he gave me to learn from him and our peers throughout those years in his Lab. His assistance and guidance are immeasurable to my success not only with this work, but as professional.

Second, but not less important I thank my parents and my brother for their unconditional support and patience, without it I would never be able to reach anything. I thank all my family as well, who made a study in a good university possible, especially my aunts Eunice and Gilvane. Also, I thank all the friends I made it during this time, who helped me and shared happiness and hard times.

Most importantly, I would like to thank God for His Grace and Provision through this journey, who comforted my family despite the distance and gave me strength to carry on no matter what challenge I faced.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent investigations in Big Data and Internet of Things (IoT) pointed out that the data cube relational operator and the Online Analytic Processing (OLAP) technology are being redesigned for new challenges (Cuzzocrea et al. (2013); Cuzzocrea (2015a); Cuzzocrea et al. (2016)), so volume, data type variety and update speed continue to be hard problems and if they appeared together in a business domain we are faced with an open problem in data cube algorithms literature.

Since the seminal paper of Gray et al. (1997), efficient data cube approaches are reducing storage and runtime impacts to compute full or partial data cubes. There are cube solutions for different data types, including traditional or alphanumeric data cubes (Sismanis et al. (2002); Xin et al. (2007); Lima and Hirata (2011)), spatial data cubes (Bimonte et al. (2006); Moreno et al. (2009); Bimonte et al. (2011)) , text data cubes (Lin et al. (2008); Zhang et al. (2009); Souza et al. (2017)), graph or network data cubes (Zhao et al. (2011); Wang et al. (2014); Benatallah et al. (2016)), RFID or stream data cubes (Gonzalez et al. (2006); Liu et al. (2011)) and image data cubes (Jin et al. (2010)). Besides many approaches for different data types, promising high performance computer (HPC) architectures that take the advantage of Graphic Processing Units (GPU), Cluster and Grid computing or cloud environments, demonstrated that the data cube problem is also hard to be efficiently partitioned and distributed (Wang et al. (2010); Kaczmarski (2011); Moreira and Lima (2012); Zhang et al. (2014); Cuzzocrea (2015b)).

The main goal of a data cube operator is to organize data into multiple hierarchies of dimensions and measures. Dimensions are composed of attributes, so they can have multiples attribute hierarchies. The *time* dimension can be composed of *year*, *month* and *day* attributes, for instance. These attributes can build the hierarchies h_1={year, day}, h_2={month, day}, h_3={year, month, day}, and many more. Besides alphanumeric dimensions, there are *text* dimensions with topic hierarchies, *spatial* dimensions with *resolution* hierarchies and many other hierarchy types in a data cube operator.

Hierarchization is a decision making active, where the information is classified into hierarchy levels, varying from detailed information to high abstraction levels of a decision making process. Drilling down an hierarchy level indicates that the analyst wants more detailed information. Roll-up is the opposite direction. There are other data cube operations, such as slice, dice and drill-through (Han et al. (2011)). In summary, they enable different alternatives to navigate through multiple hierarchy levels, simplifying the analytical decisions.

Measures are how a data cube evaluate the attribute combinations. We can suppose a combination of attribute values in a tuple t = (*year*=2016; *month*=December; *studentName*=Robert; *grade*=6.9; *coefficient*=C). The attributes *grade* and *coefficient* are measures. *Grade*, for example, can represent an average of all grades in all courses of a school. The *coefficient* 'C' can also represent a collection of courses evaluations. Measures can be a numerical value with a statistical function SUM, AVG or an inverted index to text data cubes or the distance using roadways from a location L to a river or a dam in a spatial data cube.

A data cube has base cells and aggregate cells. Suppose an input relation R with three dimensions (A, B, and C) and the tuple, $t_1 = (a_1, b_1, c_1, m)$, where a_1, b_1, and c_1 are the values for each dimensional attribute and m is a numerical value representing a measure value of t_1. In this example, each dimension has a unique attribute, so there is only one possible hierarchy. Given R, a full data cube has eight tuples representing all possible aggregations: t_1, $t_2 = (a_1, b_1, *, m)$, $t_3 = (a_1, *, c_1, m)$, $t_4 = (*, b_1, c_1, m)$, $t_5 = (a_1, *, *, m)$, $t_6 = (*, b_1, *, m)$, $t_7 = (*, *, c_1, m)$, and $t_8 = (*, *, *, m)$, where the asterisk (*) denotes a wildcard representing any value that a dimensional attribute can assume on the data cube. Generally speaking, a data cube computed from the relation R with three dimensions (A, B, and C), three attributes (a_1, b_1, c_1) and cardinality $C_A = C_B = C_C = 1$, indicating the number of unique values that each dimension can assume, can have 8 or $(C_A{+}1)\times(C_B{+}1)\times(C_C{+}1)$ tuples. In this example, t_1 is a base cell and $t_2, t_3, t_4, t_5, t_6, t_7, t_8$ are aggregate cells. The cardinalities C_A, C_B, C_C are from the three attributes of dimensions A, B and C, respectively.

If we consider relation $ABCD$ instead of relation ABC, and $C_A = C_B = C_C = C_D = 2$, there can be 16 $ABCD$ base cells and 81 aggregate cells that must be calculated in a full data cube. As we can see, the data cube operator has exponential complexity in terms of runtime and memory consumption as the dimensions increase linearly. High cardinality and huge volume of tuples in R turn the problem harder, so the Big Data requirements are demanding redesigns of data cube algorithms for indexing, querying, and updating.

The HPC literature in data cube is starting to adopt GPU cards to speedup query responses with low cost, since most recent cards bypassed 1k cores at 1GHz each, the memory capacity is increasing rapidly and current off-the-shelf PC motherboards can handle up to four cards. Unfortunately, some improvements achieved are restricted to data cubes with low dimensionality and some of them completely stored in GPU memory (Lauer et al. (2010); Wittmer et al. (2011); Wang and Zhou (2012)), which is not suitable for Big Data.

Other approaches partially store data cubes in GPU Riha et al. (2011); Malik et al. (2012), so CPU working-memory (RAM, for instance) is adopted in conjunction, but both CPU and GPU data structures are not designed for constant updates and there is always the overhead of multiple CPU-to-GPU data transfers. No cluster of GPU cards was used to index, query or update the data cube and this approach idea can achieve high speedup if CPU-GPU are used together and efficiently to handle huge volume of data. Another limitation in the GPU data cube literature is that most of the approaches implement equality operators and few range operators in their queries, like between, greater than, less than, some, similar and others. Finally, no related work innovates in supporting complex holistic measures, like MODE, RANK, many clustering operations.

1.1 Goal

Given the limitations described above, this work presents the first data cube approach to index, update and query multidimensional data over clusters of multicore-CPUs and multiple GPU cards, named JCL-GPU-Cubing. It's designed for huge volume of data and it supports recurrent updates. Holistic measures, like MODE, RANK, VARIANCE, TOP-K and many more are also supported. The alternative solution JCL-GPU-Cubing is classified as RAM-only, therefore no external memory is considered. Instead, several private main memories are used to achieve high storage capacity.

The specific goals are:

1. The design and implementation of a CPU version to index, update and query huge data cubes in multicore-CPU clusters which is used as a baseline version. Both CPU and GPU approaches adopted a third-party middleware for the distribution of tasks and data, their scheduling, their communication using several network protocols, their capacity to support faults and so forth. We considered JCL (Almeida et al. (2016)) as the underling solution due to its simplicity in terms of development and deployment. It scales well, has a distributed version of the Map Java Interface, which is very familiar to Java community, and finally it runs over small platforms, including Android devices (de Resende et al. (2017));

2. The experiments and evaluations of the CPU version using traditional alphanumeric and textual databases. Very large datasets, some of them high dimensional, and update tests are considered;

3. The design and implementation of a CPU-GPU version;

4. The experiments and evaluations of the GPU version against the CPU version with the same datasets and under the same cluster configurations. Furthermore, comparative tests with related work to corroborate the scalability results against alternatives running over a single device with a single GPU card.

1.2 Hypothesis

This work introduces a new HPC data cube approach that uses GPU devices as accelerators, therefore it must be not only faster than its CPU version, but also faster than its counterparts in the literature. These results are unique in the data cube topic, thus in database computer science area, proving that the utilization of GPU can scale OLAP technology also when clusters of multicore-CPU and GPU cards are adopted. Big Data involves huge volume of data and recurrent updates, so this approach reduces a bit more the gap of an OLAP tool to address Big Data problems.

The rest of this work is organized as follows: Chapter 2 discusses the related work about GPU as accelerators in data cube literature; Chapter 3 details the architecture and design of our solution; Chapter 4 presents the experimental results and evaluations; Chapter 5 concludes the work.

Chapter 2

Related Work

In this chapter, we describe the literature about data cubes over a single device with one or few GPUs. In all the works, the CPU memory system is unique and shared among the cores, so there is what is called a private memory system. A cluster is a group of devices that follows the private CPU memory system explained before, providing transparencies for users, like a distributed and shared memory system abstraction.

The related work was evaluated according to the following requirements obtained from the literature Cuzzocrea et al. (2016); Wittmer et al. (2011); Lauer et al. (2010); Wang and Zhou (2012); Wittmer et al. (2011); Malik et al. (2012); Riha et al. (2011); Silva et al. (2015):

1. Single or multiple GPU support (**SoMG**): is important since even off-the-shelf PC motherboards support up to four cards nowadays;

2. Multicore or cluster based deployment (**MCD**): is highlighted since distributed alternatives are quite uncommon in the literature, but distributed computing becomes the last resort when private memory systems reach their limit;

3. Large base relations support (**LBR**): is part of the Big Data concept, thus very important today, but very often the existing approaches limit the relation size to the available GPU memory, so only few gigabytes of data can be processed;

4. CPU-GPU hybrid support (**CGHS**): useful for large relations, enabling, for instance, data cube indexing in CPU and query in both CPU-GPU;

5. Support holistic measures (**HM**): imposes update and indexing challenges in data cube literature, so how to support this measure type is fundamental;

6. Support high dimensionality (**HD**): it is a hard problem since the data cube operator grows exponentially in space and time when dimensions increase linearly. Today this kind of high dimensional data is common in biological domains, but also in many social network and entertainment scenarios like films, music, news and so forth;

7. Update support (measures, dimensions and hierarchies) (**US**): is another key requirement of Big Data concept, but not implemented by most of existing OLAP products and data cube prototypes. Recurrent updates are very common, for instance, in the stock-market domain, for stream processing demands and many other niches;

8. Complex data types support (spatial, text, stream, graph and so forth) (**CDT**): represents a Big Data requirement, but unfortunately not attended until now by products or research prototypes. There are only specific solutions for specific data types, thus no integration is presented;

9. Range query operators support (between, some, greater, similar, contains, etc.) (**RQO**): enables filters not only on measures, but also on dimensions, becoming primordial in modern multidimensional analysis and Business Intelligence.

At the end of this section there is a comparative table, summarizing the benefits and drawbacks of each work according to the previously defined requirements.

The work of Lauer et al. (2010) introduced a data structure for the data cube stored in the GPU global memory, where the attributes from the tuples are allocated contiguously in that memory, but separated by each dimension. An array of measures associated to each tuple is also stored in GPU memory. A set of indexes can be calculated from the query in a way that each thread from the GPU can operate over independent sets of tuples, avoiding memory conflicts and thread serialization. To enable the use of multiple GPUs, the algorithm needs an extra preprocessing step in which it must partition the query data equally among all cards. On the CPU, each card has it's own "host thread" that waits for the result, submitting it to another thread that aggregates the final result in a parallel reduction operation. The authors evaluated their approach using just the SUM measure during the aggregation, hence it's not clear the support to holistic measures. The experimental evaluation using multiple GPUs on a single machine obtained linear query results. Compared to CPU only versions, the response time for some queries achieved up to 42 times faster using the GPU version. The approach adopts the number of tuples in a query result to indicate if the query should be processed in CPU or in GPU, but they did not present any hybrid solution that switches from GPU to CPU and vice-versa. The number of dimensions was constrained to seven and the number of tuples to tens of millions.

The approaches presented in Riha et al. (2011) and Malik et al. (2012) implemented a hybrid strategy for the query processing. A scheduler decides if a query will run only in CPU mode or if any GPU processing will be necessary. The experiments pointed out that there is a point where the cost of the query overwhelms the cost of data transfer between CPU and GPU, thus queries that demand fewer aggregations or lower processing should be executed in CPU to avoid the overhead, similar to the suggestions of Lauer et al. (2010). To perform the scheduling it is used the hardware details and a complexity estimation for the query to create a performance model. For queries processed in GPU the relation should be completely stored on its global memory and for such it is used a storage oriented by columns.

The work Kaczmarski (2011) presents a comparison between CPU and GPU data cube algorithms. The GPU alternative transfers all the data from CPU to the GPU global memory and then performs the cube creation. Similar to other works, this data transfer is the bottleneck of the whole solution, but even with a single data transfer the aggregation phase outperformed the CPU version, being 50% faster than it. The experiments used a low dimensional base relation with 5 dimensions and multiple GPUs per machine are considered to support large relations, but no cluster deployment is proposed. For multiple GPUs and many CPU cores, the approach must scan the input data once to create a parallel execution plan in which each GPU card accesses an independent intersection of the data cube. A parallel reduction phase aggregates the final results of a multidimensional query, similar to many hybrid CPU-GPU approaches, including the JCL-GPU-Cubing approach. The experimental evaluations used just the SUM measure, thus no holistic measure was investigated.

The approach Kaczmarski and Rudny (2011) presented a compact representation of a data cube and an algorithm based on the primitives parallel scan and parallel reduction to perform queries on the GPU. The base relation is entirely stored in GPU memory and when the data cube is sparse the approach introduces data compression. The representation in GPU of the data cube is not very prone for updates, so the update of dimensions, measures, hierarchy levels or a simple new attribute value would imply the re-indexing of the data cube from scratch, which is impracticable in many online or real time domains, like financial trading, stream processing, social networks, logistic and so forth.

The work of Wang and Zhou (2012) used a linearization function responsible to map each cell of the data cube to a position P of the GPU memory, where such function has the property of being reversible, that is, with P it is possible to retrieve the attribute values of each dimension. This work considers that the data is transferred to the GPU global memory when the system receives a query. Precisely, the query is interpreted in the CPU and then transferred to the GPU, which performs the filtering, data cube creation and aggregations. The design for storing tuples was not suitable for a dimensional or tuple increase since it demands even larger vectors to store the GPU memory positions. This work investigated range multidimensional queries, precisely the query operators "greater than" and "less than" applied into dimensional attributes.

The authors in Sitaridi and Ross (2012) reinforced memory accesses conflicts and thus synchronization issues in OLAP over GPU literature. To avoid memory conflicts, it is presented an algorithm that allocates specific regions of GPU memory for each thread and then reorder the query results according to those regions. If it is detected that a conflict may occur, regions are duplicated among the threads, enabling private operations without conflicts, but introducing consistency problems. The implemented operators (JOIN and GROUP BY, specifically) achieved a speedup of only 1.2 times when compared with a baseline version. When the memory conflicts are not significant, the memory footprint and processing overhead degraded the performance. The base relation must fit entirely in the GPU global memory, an important constraint. Besides that, the data type supported is limited to 4-byte integers.

The works Zhang et al. (2012) and Zhang et al. (2014) investigated spatial-temporal aggregations using GPU processors, presenting a case study about taxi rides. The generated data cube had 10 dimensions, including pick-up latitude and longitude (spatial) and pick-up time (temporal) . The authors presented algorithms and data structures to store those data types efficiently on both GPU and CPU memories, using, for example, a 4-byte representation instead of 66-byte for date-time dimensions. The parallel implementation for GPU achieved a speedup of up to 13x if compared to the CPU version.

In Riha et al. (2013), there is a query optimization solver for hybrid memory systems that adopts information beyond the query cost estimation and data availability, i.e., it uses the current workload of each processor and the memory architecture. Experiments demonstrated an accurate performance model to estimate both the processing time and the minimum response time of a query. The GPU data cube structure stores all the data in a one-dimensional array in the global memory, performing the filtering and aggregations over this array. The GPU approach can process multiple queries in parallel since its threads manipulate private memory blocks, thus avoiding thread serialization. The best hybrid CPU-GPU solution achieved a speedup of only 1.7 times while dealing with several queries in parallel if compared with the CPU only version. The base relation must fit in GPU memory and the approach runs in a single GPU card, so large relations are not suitable.

Another CPU-GPU query scheduler is proposed by Breß et al. (2013). The authors identified a limitation in query solvers based on query response times since a scenario where one of the processors outperforms the other for all queries in terms of processing time will saturate the best processor when the others end up unused. Such scenario could degrade the overall performance, but these solutions still report that the processors allocation is optimal. To attenuate the previously described limitations it was implemented several heuristics focused on optimizing the processor scheduling in terms of workload distribution to maximize the throughput across all processors. The first phase for the heuristic is determine the best device for each query operator accounting for response time only, then it must calculate a threshold where the chosen device can be sub-optimal, but improving the current throughput. Performance evaluations achieved a speedup of 1.6 times if compared against GPU versions without such query optimization heuristics.

In Breß (2014), it is presented a detailed cost estimation procedure to evaluate a multidimensional query cost, enabling CPU or GPU query allocations dynamically. The processing cost for each query and it's estimated performance in each type of processor (GPU and CPU, respectively) are calculated and used to schedule a query to the most suitable processor type.

The cost estimation procedure uses hardware details, such as thread organization and memory architecture to infer accurate results. Performance results against MontetDB Nes and Kersten (2012) demonstrated that the presented approach could be 1.8 times faster when its CPU-GPU designs are adopted in conjunction. The work reinforced the bottleneck caused by multiple CPU-GPU data transfers, but no alternatives using multiple GPUs in a single machine or in a cluster were detailed.

Nowadays the interest in implementing efficient alternatives to build multidimensional query results has been reduced significantly. Instead, efficient scheduling strategies to allocate queries in hybrid multi-core-CPU and GPU systems became the common OLAP research interest (Riha et al. (2013); Breß et al. (2013); Breß (2014)). The queries workload partition, how to avoid multiple CPU-GPU transfers, how updates work and so forth are user responsibilities, using the literature improvements, for instance. The recent works of Karnagel and Habich (2017); Appuswamy et al. (2017) reinforced this assumption. The JCL-GPU-Cubing approach innovates in different direction, i.e., we are interested in efficient algorithms to build multidimensional queries from huge data cubes over a cluster of CPUs-GPUs devices, therefore the decision to work cooperatively CPUs-GPUs or separately is not our focus because the queries investigated manipulate huge results and consume vast amount of processor's cycles, therefore CPUs and GPUs are always used together.

Table 2.1 presents a comparison of the related work described above, summarizing the implemented requirements by each paper and the requirements our work implements and will implements upon completion as well. Each requirement can be fulfilled in three levels (✓, ✓✓ and ✓✓✓), where ✓ indicates basic implementations, ✓✓ indicates fundamental ones and ✓✓✓ indicates advanced designs. The '–' wildcard indicates that no information was found about the ability of the approach in attending the requirement.

Table 2.1: Comparison of the related work using the previously defined requirements.

Related Work	**SoMG**	**MCD**	**LBR**	**CGHS**	**HM**	**HD**	**US**	**CDT**	**RQO**
JCL-GPU-Cubing	✓✓✓	✓✓✓	✓✓	✓✓✓	✓✓✓	✓✓	✓✓	–	✓✓✓
Lauer et al. (2010)	✓✓	✓	✓	–	–	–	–	–	–
Riha et al. (2011)	✓	✓	✓	✓✓✓	–	–	–	–	–
Kaczmarski (2011)	✓✓	✓	✓	–	–	–	–	–	–
Kaczmarski and Rudny (2011)	✓	✓	✓	–	–	–	–	✓	–
Wang and Zhou (2012)	✓	✓	✓	–	–	–	–	–	✓
Sitaridi and Ross (2012)	✓	✓	✓✓	–	–	–	–	–	–
Zhang et al. (2012)	✓	✓	✓	–	–	✓	–	✓✓	–
Malik et al. (2012)	✓	✓	✓	✓✓✓	–	–	–	–	–
Riha et al. (2013)	✓	✓	✓	✓✓✓	–	–	–	–	–
Breß et al. (2013)	✓	✓	–	✓✓✓	–	–	–	–	–
Breß (2014)	✓	✓	✓	✓✓✓	–	–	–	–	–
Zhang et al. (2014)	✓	✓	✓	–	–	✓	–	✓✓	–
Karnagel and Habich (2017)	✓	–	–	✓✓✓	–	–	–	–	–
Sato and Usami (2017)	✓	✓	✓	–	–	–	–	✓	–
Appuswamy et al. (2017)	✓	✓✓	–	✓✓✓	–	–	–	–	–

According to Table 2.1, the most attended requirements are: i) single GPU support (SoMG), ii) multicore deployment (MCD) and iii) medium size base relations (LBR). The most rare requirements in the literature are: v) holistic measures support (HM), vi) high dimensionality support (HD) and vii) update support (US). The remaining requirements (CGHS, CDT and RQO), although partially attended, are equally important. The big data requirements (volume, velocity and variety, for instance) are not addressed by the GPU OLAP literature, precisely single devices deployments are not sufficient when volume increases, the recurrent updates of a data cube crash its internal representation and algorithms in all related work, and finally no work creates a unified data cube representation with dimensions, measures and hierarchies for text, spatial, stream and other data types. In the next section, we present an alternative solution to attend some of requirements stated in the beginning of this section.

Chapter 3

Development

In this chapter, we present the main components of the JCL-GPU-Cubing approach. Figure 3.1 represents the indexing core idea and Figure 3.2 illustrates the query multiple pipelines. A base relation with all tuples represent the input data. Each column of it represents a dimension attribute or a measure attribute of a data cube. There is a primary-key or tuple identification per tuple, called "id".

3.1 Indexing

The data must be stored in each device of the cluster to answer queries and for that there is an input partition strategy illustrated by the "tuples partition and split" operations occurred in the beginning of the "index" method. The partition and split operations run indefinitely, i.e., while there is tuples to be consumed from a data source. A sequential process reads the input data repeatedly and sends chunks of tuples at a time to each device of the cluster. This is done once and can take hours or even days for the first load of a huge amount of data. A base relation is defined in Figure 3.1 by dimensions (d_1, d_2, $\dots d_n$) and measures (m_1, m_2, m_3, $\dots m_n$).

The partitioned version of the base relation is indexed by each device in the "index" method. The "index" method produces what we call "partial cube". The "partial cube" is composed of three data sources: "dimensions", "measures" and "tuples". The first represents the inverted index of each tuple and consequently the inverted index of the complete base relation. A tuple t_1= $\{id_1, a_1, b_1, \dots m_1, m_2, \dots\}$ has its identification or primary-key (id_1), its dimensions (a_1, b_1, c_1, $\dots$) and measures (m_1, m_2, $\dots$), so the inverted representation of t_1, named its inverted index, can be defined as it_1=$\{$ $a_1 : id_1$, $b_1 : id_1$, $c_1 : id_1$, $\dots\}$. The second data source of "partial cube" is named "measures" and it is responsible for storing the measure values and from which tuple they come from. Finally, the third data source named "tuples" represents the tuple storage without its measures. Both "measures" and "tuples" data sources represent the base relation partitioned horizontally (tuple partition and not column partition) over a cluster.

These three data sources are sufficient for answering queries efficiently in parallel and without data redundancies. After the “index” method call, several “partial cube” representations are created in the cluster, as Figure 3.1 illustrates. An important aspect of the indexing algorithm is that there is no synchronization barriers, so even a cluster composed of multi-core CPUs can be adopted without synchronization drawbacks during the partial data cube indexing.

JCL-GPU-Cubing can handle high dimensional data cubes, but without computing full data cubes, because of the dimensionality problem. The number of tuples of the base relation and the cardinality of each attribute of each dimension also impose challenges for the full data cube construction, what reinforces our architectural design choices. The query results are created on-the-fly using partial aggregations created during the index method call. In the next section we explain the query multiple pipelines.

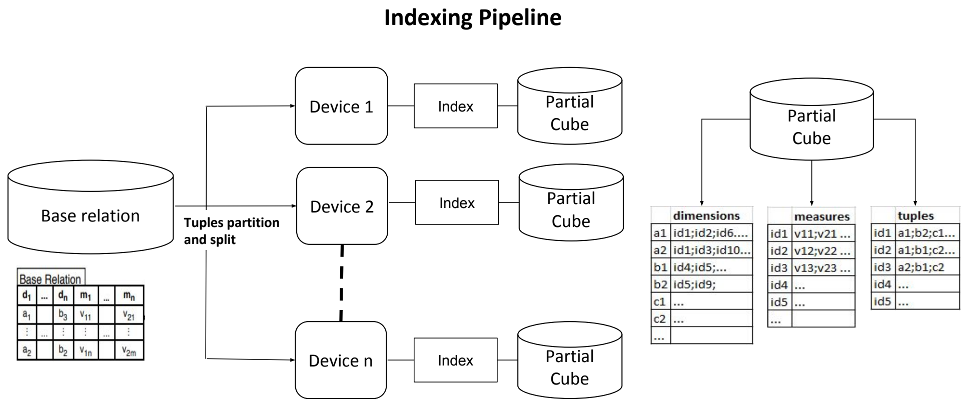

Figure 3.1: The indexing phase

3.2 Query

Once the “partial cube” is created on each device of the cluster, it is possible to perform multiple queries. The first step of a query pipeline is to define dimension filters, like “country language equals Portuguese”, “age between 18 and 45 years”, “year greater than 2000” and so forth. Normally, in a query there are multiple filters and if they are applied iteratively we have the logical operator “AND” of sub-queries in a main query sentence. Lets suppose the query q= { country language equals Portuguese; age between 18 and 45; year greater than 2000}, so each filter of the query is applied over the result set produced by the previous one. Using q as an example, the first filter (country language equals Portuguese) would be executed over the entire partial cube and the return a result set of tuples that satisfy this particular filter on the "country language" dimension, then the second filter (age between 18 and 19) is executed over this result set and returns a more strict set of tuples that now satisfy the query on the "country language" and "age" dimensions. Finally the last filter (year greater than 2000) is applied over the previous result set now evaluating its tuples on the year dimension. At the end, the final result set would satisfy all the filters from the query. Each device or each core of a device is performing all query pipeline steps on its "partial cube" representation.

After the filtering, a set of valid tuples is obtained, i.e., a set of tuples that meet the filters restrictions applying only the "AND" logical operator. The "OR" and nested "AND/OR" combinations are planed for future implementations of the presented approach. A detailed formalism of completeness and how to optimize nested "AND/OR" logical operators in a multidimensional query is planned for a master thesis or even a PhD thesis due to the complexity. With all valid tuples, it is possible to transfer all tuples to GPU to perform the "sub-cube construction" or leave them in CPU to perform the same operation. The GPU algorithm has not been concluded, so we will delay its definition and explanations until December, 2018, the deadline to present the final version of this work.

The CPU version receives all valid tuples and insert the wildcard "ALL" or "*" in all non-aggregated attribute value, which means that if a tuple t=$\{a_1, b_1, c_1, v_{11}\}$ is a valid tuple the "sub-cube construction" pipeline step will insert "*" in all the three dimensions A, B and C, creating the following new tuples t_1=$\{*, b_1, c_1, v_{11}\}$, t_2=$\{a_1, *, c_1, v_{11}\}$, t_3=$\{a_1, b_1, *, v_{11}\}$, t_4=$\{*, *, c_1, v_{11}\}$, t_5=$\{*, b_1, *, v_{11}\}$, t_6=$\{a_1, *, *, v_{11}\}$, t_7=$\{*, *, *, v_{11}\}$. The "v_{11}" represents a measure value. As we can see, this pipeline step has a high processing and storage costs, with exponential complexity in terms of number of query attributes, so in a three dimensional query there are seven new tuples plus the existing one or 2^3 tuples are the result for each valid tuple of previous pipeline step. Detailed optimizations to construct all aggregations in CPU and in GPU are part of a future work, i.e., they will be incorporated in this chapter in the second semester of 2018 in BCC 391 discipline.

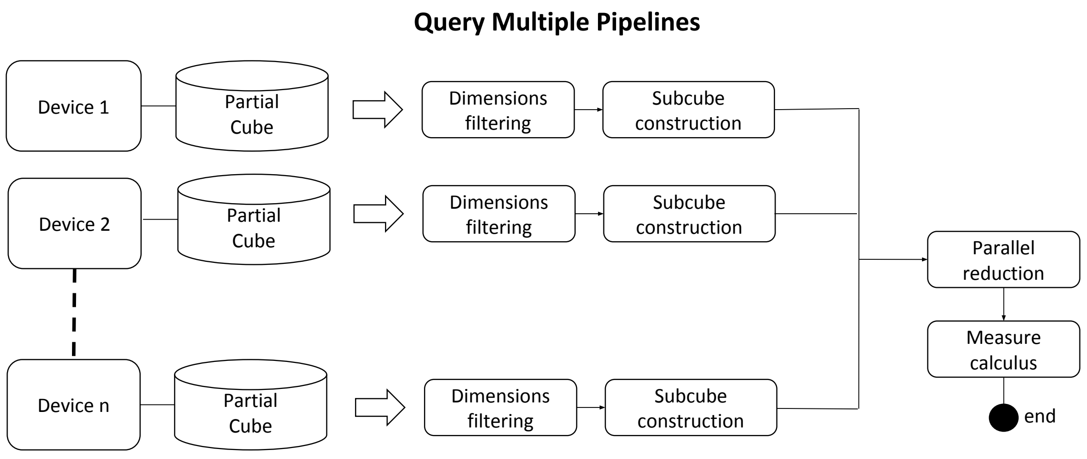

Figure 3.2: The Query phase

In the worst case, each device of a cluster receives identical tuples, producing identical "partial cubes" and consequently identical results after the query "sub-cube construction" step, so a reduction must be executed to eliminate redundancies from all devices and perform the final aggregations. This is done sequentially by a device of the cluster, so any device can perform the "parallel reduction" step of the pipeline, illustrated in Figure 3.2. Basically, this device receives all tuples and their corresponding set of measure values from the individual "partial cubes" and using the "measures" indexing structure, as Figure 3.1 illustrates. In the previous example, all t to t_7 tuples measure values (Ex. "v_{11}") produced by each device must be joined to form a final set of measure values associated with each tuple.

The final step of a query pipeline, after the "parallel reduction" step, is the "measure calculus" step, responsible to sequentially calculate the aggregated measure values using a measure function, such as AVG or COUNT or the spatial distance of two locations. After the "measure calculus" step just one representation of tuples t-t_7 exist and all measure values are finally aggregated, so the results are ready to be presented to users. Optimized versions of "parallel reduction" and "measure calculus" steps are planned and they will run distributed, i.e., instead of a single device the approach will use pairs of devices, so a cluster with 1024 devices can use 512 devices for reducing results initially, after use 256 to reduce even more. As we can see, the final operation is a sequential reduction, but from two other reduced results and not all of them, so query speedup can be increased. The same idea can be done for "measure calculus" pipeline step. This reduction idea in pairs is the basis of technologies like Map-Reduce (Dean and Ghemawat (2008)).

3.3 Discussions

In this chapter we introduce the initial ideas of an approach named JCL-GPU-Cubing, developed to achieve high speedups while answering complex multidimensional queries from huge data cubes and over CPU-GPU clusters. We present two solutions, one for indexing and other for query multidimensional data. The update service follows mainly the indexing counterpart, so we omit its explanation now, postponing it for the next semester. This work represents the first part of a more complex solution designed for clusters of multi-core CPU devices that have also GPU cards with hundreds or even thousands of cores. To conclude the work, this chapter needs:

1. Algorithms: several detailed algorithms and explanations about them must be given for almost all steps of all pipelines explained in this chapter;

2. Examples: after an algorithm explanation or during it we must introduce examples for better understanding;

3. GPU version: we must explain how the GPU works together with CPU, but also how the step "sub-cube construction" is efficiently implemented for the GPU's memory and

cores;

4. Update support: we must detail the supported type of updates on the data cube and how the indexing algorithm is changed or extended to handle the different kinds of update;

5. AND & OR logical operators: the query must be formally defined in this chapter and how AND and OR logical operators are used in a query must be defined and exemplified;

6. Distributed and sequential solutions for the "parallel reduction" and "measure calculus" steps: both must be detailed;

7. Discussions: we must introduce deep discussions about JCL-GPU-Cubing strengths and limitations, always comparing it against the state-of-art in OLAP, precisely against the research frontier in GPU accelerators to speedup data cube queries.

Chapter 4

Experiments

In this chapter we present the initial experimental evaluations of a CPU-only distributed version that will be used as a baseline solution. We first present the setup of the test. Next, we evaluated the impact of an heterogeneous cluster to the scalability, adding devices with different processing and memory capacities. At the end of this chapter, a discussion enumerates the improvements to be done in order to complete the work.

4.1 Setup

The environment consists in an heterogeneous cluster with 6 nodes of various processing and memory capacities. The devices are interconnected via an standard gigabit Ethernet switch. We tested such configuration with two base relations: BR1) with 520,000 tuples, 8 dimensions and 5 measures and BR2) with 1 million tuples, with 8 dimensions and 2 measures.

As we discussed on Chapter 3, we do not build the full data cube from the base relation; instead, we only build a partial data cube that corresponds to each query with the dimensions and measures expressed on it. Therefore, the data cubes for both queries have 4 dimensions and 1 measure. The queries are depicted below on an SQL-like language.

The query executed over BR1 was:

```
Q2: MAX total_score
    WHERE Cargo CONTAINS ‘or’ AND Loja CONTAINS ‘Shop’
     AND username CONTAINS ‘2’;
```

The query executed over BR2 was:

```
Q1: SELECT MAX PrecoUnitario
    WHERE Categoria > 5 AND Pais = ‘Brasil’ AND Cidade STARTSWITH ‘Rio’
     AND Produto ENDSWITH ‘s’;
```

The devices’ configurations are as follows (from the *best* to *worst* device).

1. Windows 7, Intel(R) Core(TM) i7-4790 @ 3.60GHz, 8 cores, 16GB of RAM;

2. Windows 7, Intel(R) Core(TM) i5-2500 @ 3.30GHz, 4 cores, 32GB of RAM;

3. Ubuntu 16.04, Intel(R) Xeon(TM) @ 3.00GHz, 4 cores, 16GB of RAM;

4. Ubuntu 16.04, Intel(R) Core(TM) @ i5-2400 3.10GHz, 4 cores, 8GB of RAM;

5. Fedora 23, Intel(R) Xeon(R) E5405 @ 2.00GHz, 8 cores, 16GB of RAM;

6. Ubuntu 16.04, Intel(R) Xeon(R) E5310 @ 1.60GHz, 4 cores, 8GB of RAM;

We evaluated the index and the query pipelines in terms of total time, i.e., the time elapsed to index or query are presented, so network, stack and other times are omitted. Each query was tested 3 times over each cluster configuration explained before and the final values used on graphics of this chapter are the average of the 3 execution times obtained, presenting also the standard deviation on the plots. All cluster devices received the same number of tuples, so there is no partition strategy to create a base relation map or schema over the cluster according to base relation dimensional attributes skew and the devices online information about RAM utilization, CPU percentage, number of processes in the operating system stack and many more. Actually, we just adopt a circular list strategy to partition the base relation tuples among the cluster devices.

4.2 Experimental Evaluations

We started the cluster configurations from the 3 most powerful devices in terms of processing power and memory capacity, and then double the cluster size from 3 to 6. Still on the heterogeneous environment, we started with the 3 least powerful devices and then add the best 3 afterwards. On each configuration we run the queries over the two base relations.

4.2.1 Base relations BR1 and BR2 - Index

First, in Figure 4.1 it is illustrated the index linear behavior while tuples increase, an expected scenario. As seen on graph 4.1, the execution times do not vary much when starting from the 3 best devices and from the 3 worst. This can be explained by the nature of the index task: the indexing of a data base is a network and memory intensive step of our pipeline, since the chunks of base relation need to be transmitted across the cluster and then copied to different local data structures. Therefore, if it is a base relation not significantly big, the processing capacities are not so relevant for index in JCL-GPU-Cubing. As expected, when we double the cluster size, we now have each cluster device with smaller parts of the base relation, therefore the execution times drops, since the workload is better distributed across the cluster.

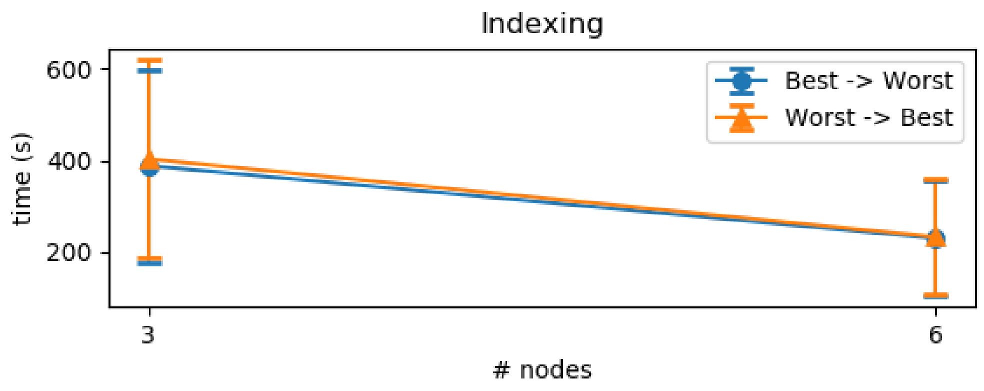

Figure 4.1: Indexing execution times starting from the best hardware and then starting from the worst on BR1

Figure 4.2 illustrates the base relation $BR2$ index operation that is almost twice as big as the previous one. Figure 4.2 shows that the JCL-GPU-Cubing index performance stays close if we vary the devices we start with. The difference is noticeable, but that can be explained by the memory bandwidth of the devices, as discussed before the index step is memory intensive, so if we have more volume of tuples the best devices tends to perform better when transferring data between the multiple data structures. When we double the number of cluster devices the performance gains are more expressive, much due to the base relation being better partitioned across the cluster.

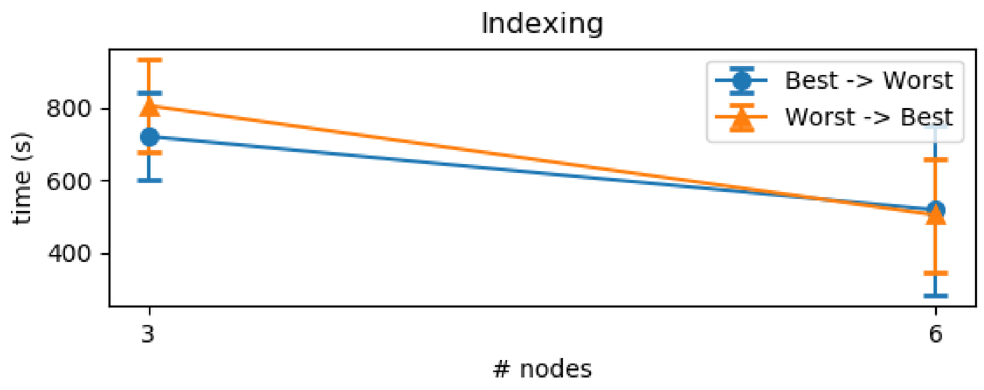

Figure 4.2: Indexing execution times starting from the best hardware and then starting from the worst on BR2

4.2.2 Base relations BR1 and BR2 - Query

In this experiment the Q1 is performed over BR1 and it is evaluated the scalability of the solution then we double the number of devices on the cluster, but keeping the input fixed.

Figure 4.3 illustrates that the differences from starting from the best and worst devices is noticeable. This occurs because the query involves the data filtration, cube construction and measure calculation, therefore the query operation is much more processing intensive than the index one. This way, starting from the best devices is the best cluster configuration. When we double the number of cluster devices, the difference from starting strategies is more significant since there are much more data and consequently bigger partial data cubes to be queried.

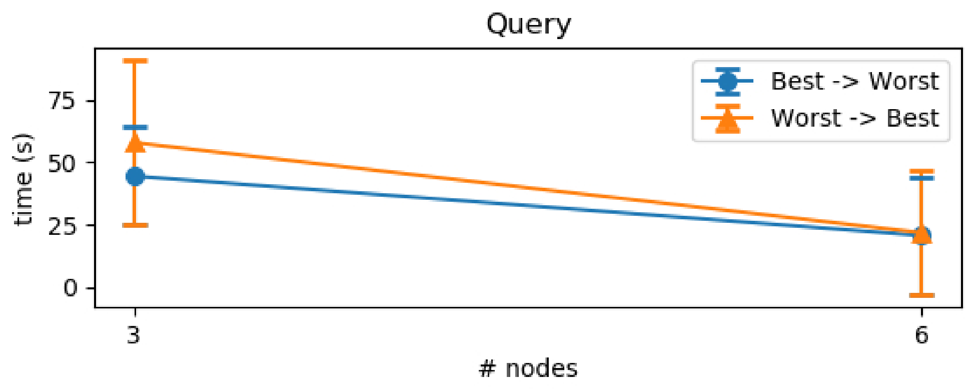

Figure 4.3: Query execution times starting from the best hardware and then starting from the worst on BR1

As expected, the query performance is much benefited by the use of the devices with best hardware configurations (Figure 4.4). When we double the cluster size, improving significantly the performance capacity, the execution times dropped almost half if compared with cluster configurations started from the worst devices.

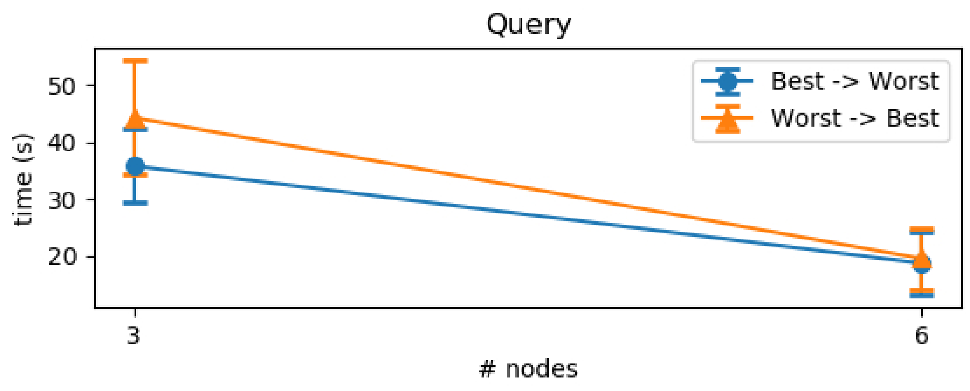

Figure 4.4: Query execution times starting from the best hardware and then starting from the worst on BR2

4.3 Discussions

In this chapter we presented the initial experimental evaluations of the JCL-GPU-Cubing approach. The current implementation uses only the CPU processing of a cluster to perform both the index and query data cube operations. This implementation, as mentioned before, will be used as a baseline against the GPU implementation of the query pipeline. This way, to conclude the experimental evaluations for the final work we need:

1. Test JCL-GPU-Cubing approach against synthetic base relations with different number of tuples, measures, dimensions, cardinalities and skews. Base relations with hundred of millions of high dimensional tuples should be supported;

2. Test JCL-GPU-Cubing with real base relations from different domains;

3. Test JCL-GPU-Cubing against a CPU baseline version;

4. Test GPU version against the related work;

5. Test the current circular list tuple partition against a tuple partition strategy that considers the RAM capacity, the number of cores and CPU clock of each cluster device;

6. Perform a deep discussion about all results of all experimental evaluations, highlighting the strengths and drawbacks of JCL-GPU-Cubing.

Chapter 5

Conclusion

This work presents the initial ideas of a solution to solve a fundamental problem in OLAP literature and consequently for the data cube relational operator: how to reduce complex multidimensional queries response times? The problem becomes even worse if recurrent updates in the input data is supported and if there is a huge volume of high dimensional data to be analyzed.

This work presents an alternative solution to solve the problem stated before. We are interested in using GPU cards to speedup queries. This idea has been done before, but using a single device and two GPU cards at most. We are interested in using cluster deployments, so multiple private memories must be considered. Many related work require the base relation totally stored in GPU memory, which is unrealistic for large base relations. Sometimes they do not support holistic measures or range queries or hybrid CPU-GPU benefits. This work mitigates most of these limitations, precisely: i) huge volume of data; ii) high dimensionality; iii) holistic measures; iv) range queries; v) hybrid CPU-GPU solution; vi) cluster version and not only multi-core version to speedup queries.

Experimental results demonstrated that the CPU version scales when both the input data and the number of devices increase. These preliminaries results reinforce our hypothesis that GPU clusters can speedup query response times. We plan future tests, including comparative ones with GPU and CPU versions. More experiments with synthetic and real data are necessary. The data cube proprieties, i.e., its number of dimensions, tuples, its cardinality and skew must be varied for a better understanding about JCL-GPU-Cubing benefits and limitations.

The detailed planning for the conclusion of this work is illustrated in Table 5.1

Table 5.1: Planning of the remaining tasks to conclude this work

Task	**Date**
GPU implementation of the query pipeline	August-September 2018
Distributed and sequential solutions of the “parallel reductions” and “measure calculus” steps	September 2018
Update Support	September-October 2018
Support for AND & OR logical operators and detailed explanations	October 2018
Detailed explanations of the algorithms	October 2018
Examples using the algorithms	October 2018
Tests with more realistic and analyzed data	November 2018
Experimental Evaluation of the GPU implementation	November 2018
Comparative experiments with related work	November-December 2018
Discussions about our final solution against the related work	December 2018
Finish the monograph and submit a paper with our findings	December 2018

Bibliography

Almeida, A. L. B., Silva, S. E. D., Nazaré Jr, A. C., and de Castro Lima, J. (2016). Jcl: A high performance computing java middleware. In *ICEIS (1)*, pages 379–390.

Appuswamy, R., Karpathiotakis, M., Porobic, D., and Ailamaki, A. (2017). The case for heterogeneous htap. In *8th Biennial Conference on Innovative Data Systems Research.*

Benatallah, B., Motahari-Nezhad, H. R., et al. (2016). Scalable graph-based olap analytics over process execution data. *Distributed and Parallel Databases*, 34(3):379–423.

Bimonte, S., Tchounikine, A., and Miquel, M. (2006). Geocube, a multidimensional model and navigation operators handling complex measures: Application in spatial olap. In *ADVIS*, pages 100–109. Springer.

Bimonte, S., Tchounikine, A., Miquel, M., and Pinet, F. (2011). When spatial analysis meets olap: Multidimensional model and operators. *Exploring Advances in Interdisciplinary Data Mining and Analytics*, pages 249–277.

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

Breß, S., Siegmund, N., Bellatreche, L., and Saake, G. (2013). An operator-stream-based scheduling engine for effective gpu coprocessing. In *East European Conference on Advances in Databases and Information Systems*, pages 288–301. Springer.

Cuzzocrea, A. (2015a). Aggregation and multidimensional analysis of big data for large-scale scientific applications: Models, issues, analytics, and beyond. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, SSDBM '15, pages 23:1–23:6, New York, NY, USA. ACM.

Cuzzocrea, A. (2015b). Data warehousing and olap over big data: a survey of the state-of-the-art, open problems and future challenges. *International Journal of Business Process Integration and Management*, 7(4):372–377.

Cuzzocrea, A., Bellatreche, L., and Song, I.-Y. (2013). Data warehousing and olap over big data: Current challenges and future research directions. In *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*, DOLAP '13, pages 67–70, New York, NY, USA. ACM.

Cuzzocrea, A., Simitsis, A., and Song, I.-Y. (2016). Big data management: New frontiers, new paradigms. *Information Systems*.

de Resende, J. E. E., de Souza Cimino, L., Silva, L. H. M., Rocha, S. Q. S., de Oliveira Correia, M., Monteiro, G. S., de Souza Fernandes, G. N., Almeida, S. G. M., Almeida, A. L. B., de Aquino, A. L. L., and de Castro Lima, J. (2017). Jcl android : Uma extensão iot para o middleware hpc jcl. In *Brazilian Symposium on Computing Systems Engineering*.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Gonzalez, H., Han, J., and Li, X. (2006). Flowcube: constructing rfid flowcubes for multi-dimensional analysis of commodity flows. In *Proceedings of the 32nd international conference on Very large data bases*, pages 834–845. VLDB Endowment.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., and Pirahesh, H. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53.

Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Jin, X., Han, J., Cao, L., Luo, J., Ding, B., and Lin, C. X. (2010). Visual cube and on-line analytical processing of images. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 849–858. ACM.

Kaczmarski, K. (2011). Comparing gpu and cpu in olap cubes creation. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 308–319. Springer.

Kaczmarski, K. and Rudny, T. (2011). Molap cube based on parallel scan algorithm. In *Advances in Databases and Information Systems*, pages 125–138. Springer.

Karnagel, T. and Habich, D. (2017). Heterogeneous placement optimization for database query processing. *it-Information Technology*, 59(3):117–123.

Lauer, T., Datta, A., Khadikov, Z., and Anselm, C. (2010). Exploring graphics processing units as parallel coprocessors for online aggregation. In *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP*, pages 77–84. ACM.

Lima, J. C. and Hirata, C. M. (2011). Multidimensional cyclic graph approach: Representing a data cube without common sub-graphs. *Information Sciences*, 181(13):2626–2655.

Lin, C. X., Ding, B., Han, J., Zhu, F., and Zhao, B. (2008). Text cube: Computing ir measures for multidimensional text database analysis. In *2008 Eighth IEEE International Conference on Data Mining*, pages 905–910. IEEE.

Liu, M., Rundensteiner, E., Greenfield, K., Gupta, C., Wang, S., Ari, I., and Mehta, A. (2011). E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 889–900. ACM.

Malik, M., Riha, L., Shea, C., and El-Ghazawi, T. (2012). Task scheduling for gpu accelerated hybrid olap systems with multi-core support and text-to-integer translation. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1987–1996. IEEE.

Moreira, A. A. and Lima, J. C. (2012). Full and partial data cube computation and representation over commodity pcs. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 672–679. IEEE.

Moreno, F., Arango, F., and Fileto, R. (2009). Extending the map cube operator with multiple spatial aggregate functions and map overlay. In *Geoinformatics, 2009 17th International Conference on*, pages 1–7. IEEE.

Nes, S. I. F. G. N. and Kersten, S. M. S. M. M. (2012). Monetdb: Two decades of research in column-oriented database architectures. *Data Engineering*, 40.

Riha, L., Malik, M., and El-Ghazawi, T. (2013). An adaptive hybrid olap architecture with optimized memory access patterns. *Cluster computing*, 16(4):663–677.

Riha, L., Shea, C., Malik, M., and El-Ghazawi, T. (2011). Task scheduling for gpu accelerated olap systems. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, pages 107–119. IBM Corp.

Sato, H. and Usami, T. (2017). Skycube-tree based query processing in olap skyline cubes. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, page 64. ACM.

Silva, R. R., Hirata, C. M., and de Castro Lima, J. (2015). A hybrid memory data cube approach for high dimension relations. In *ICEIS (1)*, pages 139–149.

Sismanis, Y., Deligiannakis, A., Roussopoulos, N., and Kotidis, Y. (2002). Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 464–475. ACM.

Sitaridi, E. A. and Ross, K. A. (2012). Ameliorating memory contention of olap operators on gpu processors. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, pages 39–47. ACM.

Souza, A. N. P., Fortes, R. S., and Lima, J. C. (2017). Dynamic topic hierarchies and segmented rankings in textual olap technology. *Journal of Convergence Information Technology (JCIT)*, 12(1):1–17.

Wang, G. and Zhou, G. (2012). Gpu-based aggregation of on-line analytical processing. In *Communications and Information Processing*, pages 234–245. Springer.

Wang, Y., Song, A., and Luo, J. (2010). A mapreducemerge-based data cube construction method. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 1–6. IEEE.

Wang, Z., Fan, Q., Wang, H., Tan, K.-L., Agrawal, D., and El Abbadi, A. (2014). Pagrol: parallel graph olap over large-scale attributed graphs. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 496–507. IEEE.

Wittmer, S., Lauer, T., and Datta, A. (2011). Real-time computation of advanced rules in olap databases. In *Advances in Databases and Information Systems*, pages 139–152. Springer.

Xin, D., Han, J., Li, X., Shao, Z., and Wah, B. W. (2007). Computing iceberg cubes by top-down and bottom-up integration: The starcubing approach. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):111–126.

Zhang, D., Zhai, C., and Han, J. (2009). Topic cube: Topic modeling for olap on multidimensional text databases. In *SDM*, volume 9, pages 1124–1135. SIAM.

Zhang, J., You, S., and Gruenwald, L. (2012). High-performance online spatial and temporal aggregations on multi-core cpus and many-core gpus. In *Proceedings of the fifteenth international workshop on Data warehousing and OLAP*, pages 89–96. ACM.

Zhang, J., You, S., and Gruenwald, L. (2014). Parallel online spatial and temporal aggregations on multi-core cpus and many-core gpus. *Information Systems*, 44:134–154.

Zhao, P., Li, X., Xin, D., and Han, J. (2011). Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864. ACM.