cgmOLAP: Efficient Parallel Generation and Querying of Terabyte Size ROLAP Data Cubes

Y. Chen & A. Rau-Chaplin
Dalhousie University
Halifax, Canada

F. Dehne
Carleton University
Ottawa, Canada

T. Eavis
Concordia University
Montreal, Canada

D. Green & E. Sithirasenan
Griffith University
Brisbane, Australia

Abstract

We present the cgmOLAP server, the first fully functional parallel OLAP system able to build data cubes at a rate of more than 1 Terabyte per hour. cgmOLAP incorporates a variety of novel approaches for the parallel computation of full cubes, partial cubes, and iceberg cubes as well as new parallel cube indexing schemes. The cgmOLAP system consists of an application interface, a parallel query engine, a parallel cube materialization engine, meta data and cost model repositories, and shared server components that provide uniform management of I/O, memory, communications, and disk resources.

1 Overview

We present the cgmOLAP server (http://cgmOLAP.cgmlab.org), developed as part of the PANDA project (http://panda.cgmlab.org). The cgmOLAP server employs parallel processing techniques to support a highly scalable ROLAP data cube system. To the best of our knowledge it is the first fully functional parallel OLAP system able to build data cubes at a rate of more than 1 Terabyte per hour.

For a given raw data set, R, with N records and d attributes (dimensions), the pre-computation of the full data cube (the set of all 2^d possible views) or a partial data cube (a subset of all 2^d possible views) supports the fast execution of subsequent OLAP queries [7]. The size of data cubes can be massive. In the Winter Corporation's report (www.wintercorp.com/vldb/), the largest three DSS databases exceed 20 Terabytes in size. Data cubes can be one or two orders of magnitude larger than the input databases. Parallel processing can provide two key ingredients for dealing with the data cube size: increased computational power through multiple processors and increased I/O bandwidth through multiple parallel disks (e.g. [1, 3, 4, 8]). Compared to previous systems, our cgmOLAP server has the following **distinguishing characteristics**:

1. **Fully parallel.** The cgmOLAP server is fully parallel and has been designed from the ground up to efficiently exploit the computational power of inexpensive, shared-nothing, distributed memory clusters.
2. **Memory hierarchy adaptive.** All of the key algorithms that make up the cgmOLAP server are designed to be both cache friendly and capable of running fully in external memory. No requirement that even data stored on a single processor will fit in memory. Detailed engineering of I/O managers to manage local disk subsystems.
3. **Highly tunable.** All of the key algorithms that make up the cgmOLAP server are driven off an explicit cost model. This approach supports both hardware platform portability and application self tuning.
4. **Highly scalable.** The cgmOLAP server is highly scalable in terms of dimensions, processors, and input records. All of the key parallel algorithms for data cube generation and query processing exhibit close to linear (optimal) speedup for as many as 32 processors. Optimal speedup for 32 processors was not observed for any previous parallel methods [6, 8, 9, 10]. In addition, because of its shared-nothing approach, cgmOLAP is also a significant improvement with respect to the I/O bandwidth.

2 cgmOLAP For Full & Partial Data Cubes

We first consider the construction of data cubes without data reduction (in contrast to iceberg cubes). The global structure of our parallel data cube construction algorithm for shared-nothing multiprocessors [3] consists of d iterations $i = 1 \ldots d$. In iteration i, the i-subcube DC_i is created. At first, each processor j computes a local subcube DC_{ij} for its local data set. Then, we merge the DC_{ij} to obtain the correct i-subcube DC_i. The merge operation is the critical part of our method [3]. Our algorithm can be applied to both full and partial data cube construction, where only a subset of the 2^d possible views is to be created [5]. Good data partitioning is the key factor in obtaining good performance. We have devised a dynamic data partitioning scheme called "pivoting". A carefully selected set of pivots is chosen to partition the data and ensure minimum data movement during the merging of the DC_{ij}. This dynamic data partitioning scheme adapts to both the current data set and the performance parameters of the parallel machine. Using this scheme, data cube generation tasks involving millions of rows of input, that take days to perform on a single processor machine, can be completed in just hours on a 32 processor cluster. The optimized data partition scheme exhibits optimal, linear, speedup for full cube generation on as many as 32 processors, as well as excellent sizeup and scaleup behavior. For example, for a fact table with 16 million rows and 8 attributes, our parallel data cube generation method achieves close to optimal speedup for 32 processors, generating a full data cube in under 3 minutes. For a fact table with 256 million rows and 8 attributes, our parallel method achieves optimal speedup for 32 processors, generating a full data cube consisting of ≈ 7 billion rows (200 Gigabytes) in under 37 minutes.

3 cgmOLAP For Iceberg Cubes

For iceberg cubes, aggregate values are only stored if they have a certain, user specified, minimum support. In [2] we introduced the "Pipe 'n Prune" (PnP) operator for parallel and external memory iceberg cube computation. The PnP operator is part of cgmOLAP. The novelty of our method (illustrated in Figure 1) is that it completely interleaves a top-down piping approach for data aggregation with bottom-up Apriori data pruning. The idea behind PnP is to fully integrate data aggregation via top-down piping with bottom-up Apriori pruning. For a group-by v, the PnP operator performs two steps: (1) It builds all group-bys v' that are a prefix of v through one single sort/scan operation with iceberg cube pruning. (2) It uses these prefix group-bys to perform bottom-up Apriori pruning for new group-bys that are starting points of other piping operations. The PnP operator is applied recursively until all group-bys of the iceberg cube have been generated. An example of a 5-dimensional *PnP Tree* depicting the entire process for a 5-dimensional iceberg cube query is shown in Figure 1. A particular strength of PnP is that it is very efficient for *all* of the following scenarios: (1) Sequential iceberg cube queries. (2) External memory iceberg cube queries. (3) Parallel iceberg cube queries on shared-nothing PC clusters with multiple disks. PnP performs very well for both dense *and* sparse data sets and it scales well, providing linear speedup for larger number of processors. In [10] Ng et. al. observe for their parallel iceberg cube method that "the speedup from 8 processors to 16 processors is below expectation" and attribute this scalability problem to scheduling and load balancing issues. Our analysis shows that PnP solves these problems and scales well for at least up to 32 processors. For example, for a fact table with 10 million rows and 10 attributes, our PnP implementation builds the iceberg cube of 363984 rows (8.8 MB) on 16 processors in less than 3 minutes.

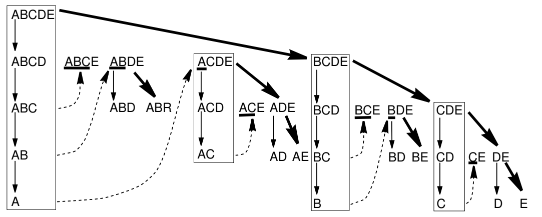

Figure 1. A PnP Tree. (Plain arrow: Top-Down Piping. Dashed Arrow: Bottom-up Pruning. Bold Arrow: Sorting.)

References

[1] Chen, Dehne, Eavis, Rau-Chaplin. Parallel ROLAP data cube construction on shared-nothing multiprocessors. *IPDPS*, 2003.

[2] Chen, Dehne, Eavis, Rau-Chaplin. PnP: Parallel and external memory iceberg cube computation. *ICDE*, 2005.

[3] Chen, Dehne, Eavis, Rau-Chaplin. Building large ROLAP data cubes in parallel. *IDEAS*, 2004.

[4] Dehne, Eavis, Rau-Chaplin. Parallelizing the datacube. *Distributed and Parallel Databases*, 11(2), 2002.

[5] Dehne, Eavis, Rau-Chaplin. Computing partial data cubes. *HICSS-37*, 2004.

[6] Goil, Choudhary. A parallel scalable infrastructure for OLAP and data mining. *IDEAS*, 1999.

[7] Gray et.al. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1), 1997.

[8] Lu, Yu, Feng, Li. Fully dynamic partitioning: Handling data skew in parallel data cube computation. *Distributed and Parallel Databases*, 13, 2003.

[9] Muto, Kitsuregawa. A dynamic load balancing strategy for parallel datacube computation. *DOLAP*, 1999.

[10] Ng, Wagner, Y. Yin. Iceberg-cube computation with PC clusters. *SIGMOD*, 2001.