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

Ying Chen
Dalhousie University
Halifax, Canada
ychen@cs.dal.ca

Frank Dehne
Griffith University
Brisbane, Australia
www.dehne.net

Todd Eavis
Concordia University
Montreal, Canada
eavis@cs.concordia.ca

Andrew Rau-Chaplin
Dalhousie University
Halifax, Canada
www.cs.dal.ca/∼arc

Abstract

In this demo 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. The cgmOLAP demo system will be running on two 32 processor Linux-based clusters, one located in Canada the other in Australia. Our demonstration interface consists of three parts: (1) a cube specification panel that allows attendees to initiate the parallel generation of full, partial, or iceberg cubes (2) a query specification panel that allows attendees to initiate parallel range, rollup, drill-down, slice, dice and pivot queries (3) a performance monitoring panel which supports the visualization of cube generation and querying performance parameters such as rows generated, or queries evaluated, per second.

1 Introduction

In this demo we present the cgmOLAP server, developed as part of the PANDA project [1] at Dalhousie, Concordia, Carleton and Griffith universities. The cgmOLAP server employs parallel processing techniques to support a highly scalable ROLAP data cube system. The size of a single data cube query can be massive. cgmOLAP 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), a view is constructed by an aggregation of R along a subset of attributes. As proposed in [11], 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. The size of data cubes can be massive. In the Winter Corporation's report [15], the largest three DSS databases exceed 20 Terabytes in size. More importantly, it is expected that as of 2005, the storage requirements of more than 40% of production data warehouses will exceed one Terabyte [8].

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. [3, 5, 6, 10, 12]). In particular, 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. **Scalability.** The cgmOLAP server is highly scalable in terms of dimensions, processors, and input records.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

All of the key parallel algorithms for data cube generation and query processing exhibit close to linear (optimal) speedup.

2 The cgmOLAP Architecture and Hardware Platform.

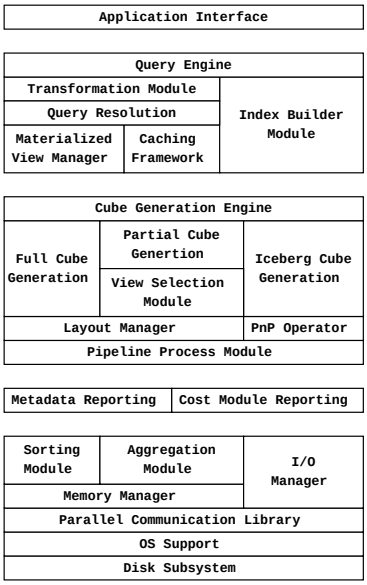

Figure 1: cgmOLAP software architecture.

Key system components include:

1. **Application interface.** Provides the application interface to the cgmOLAP system.
2. **Parallel query engine.** Supports parallel rollup, drill-down, slice, dice, and pivot queries as well as d-dimensional range search.
3. **Parallel cube materialization engine.** Supports parallel generation of full, partial and iceberg cubes.
4. **Meta data and cost model repositories.** Repositories for meta data describing original data sets and materialized cubes, as well as hardware configuration and system performance parameters that drive the algorithmic cost models.
5. **Shared server components.** These components provide uniform management of I/O, memory, communications, and disk resources.

A particular feature of cgmOLAP is that it shows very high performance on low cost, shared nothing, commodity cluster architectures. A diagram of such an architecture is given in Figure 2. Consider a storage node as shown in Figure 2 that holds the initial, possibly very large, raw data set R on a disk array. cgmOLAP fully distributes the data set R over the p local disks of processors P_0 to P_{p-1} in striped format as shown in Figure 3. Similarly, each view created by cgmOLAP is generated over the p local disks in striped format, using a Hilbert order space filling curve. The striped format ensures that subsequent accesses to an individual view generated by cgmOLAP can access all p local disks in parallel, providing maximum I/O bandwidth, with balanced retrieval across the disks/processors.

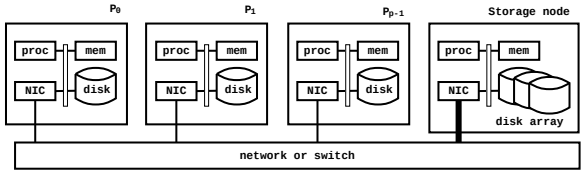

Figure 2: Shared-Nothing Multiprocessor.

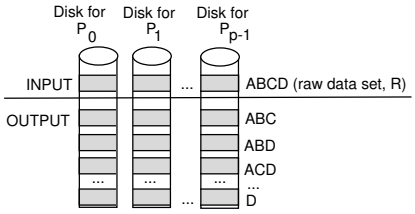

Figure 3: Parallel Disk Layout.

3 cgmOLAP For Uncompressed Data Cubes

We first consider the construction of data cubes without data reduction (in contrast to iceberg cubes [2, 9, 16] where aggregate values are only stored if they have a certain, user specified, min support). The global structure of our parallel data cube construction algorithm for shared-nothing multiprocessors [5] is illustrated in Figure 4 and consists of d iterations $i = 1 \ldots d$. In iteration i, the i-subcube

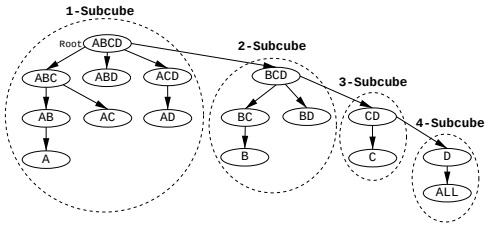

Figure 4: Subcubes of a data cube for $d = 4$.

DC_i is created in five main steps: Computing $Root_i$, computing the schedule tree T_i, optimizing the partitioning of $Root_i$ into $Root_{i1}$ ... $Root_{ip}$ over the p local disks, computing on each processor the local DC_{ij} from each $Root_{ij}$, and merging the DC_{ij} to obtain the correct i-subcube DC_i. The 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. The only difference is in the chosen schedule tree T_i. For a full cube we use Pipesort to compute T_i and for a partial cube we use a modified schedule tree construction method presented in [7].

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. We have performed an extensive performance evaluation of our new method, exploring relative speedup, scaleup, sizeup, output sizes and data skew. The optimized data partition scheme exhibited optimal, linear, speedup for full cube generation on as many as 32 processors, as well as excellent sizeup and scaleup behavior. For the planned demonstration, we will provide data sets where the response times are within the range of a few minutes. 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. We will also execute a few larger demo runs in the background. For example, 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 $\approx$ 7 billion rows (200 Gigabytes) in under 37 minutes.

4 cgmOLAP For Iceberg Cubes

For iceberg cubes (e.g. [2, 9, 16]), aggregate values are only stored if they have a certain, user specified, minimum support. In [4] 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, which is illustrated in Figure 5 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 [14] with bottom-up (BUC [2]) 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 (piping [14]) with iceberg cube pruning. (2) It uses these prefix group-bys to perform bottom-up (BUC [2]) 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 5. 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.

Our performance analysis shows that PnP performs very well for both, dense *and* sparse data sets and it scales well, providing linear speedup for larger number of processors. In [13] 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 the planned demonstration, we will provide data sets where the response times are within the range of a few minutes. 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.

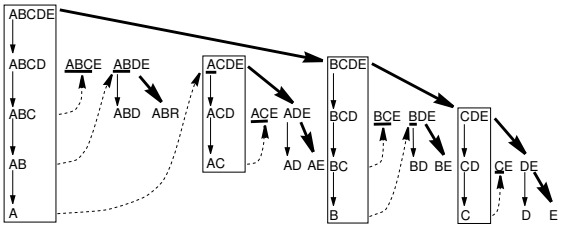

Figure 5: A PnP Tree. (Plain arrow: Top-Down Piping. Dashed Arrow: Bottom-up Pruning. Bold Arrow: Sorting.)

5 Parallel OLAP Query Processing

While the previous algorithms can be used to produce aggregated data cubes, the associated tables may still be quite large. Moreover, given the complex, multi-dimensional nature of the OLAP environment, a naive query implementation would significantly undermine the potential benefit of the materialized cubes. Therefore, it is crucial that an efficient, OLAP-centric query engine be developed to support the existing suite of cube generation algorithms.

Given the requirement for direct and intuitive multi-dimensional query functionality, a dimension-aware indexing mechanism is required. Specifically, each aggregated view is supported by an r-tree that has been "packed" using the Hilbert space filling curve. Packing not only allows more efficient storage, but it is *dimension neutral*, thereby ameliorating the performance issues often associated with other packing schemes. Our r-trees are fully parallelized, using a round robin striping mechanism to distribute individual Hilbert-ordered views to each of the p nodes of the parallel machine. The resulting framework creates a set of p partial packed r-trees, each of which contributes equally to the resolution of individual queries. Experimental analysis has shown the query load balancing error to be consistently less the 1%. We note that the collective resolution of queries is more appropriate in data warehousing environments, where queries tend to be larger, more complex, and less frequent than would be the case in OLTP settings.

Figure 6 depicts the basic query model. Essentially, the server *virtualizes* the data cube so that the user simply sees a fully materialized data cube that is indexable on all dimensions and/or dimension hierarchies. In order to maintain this transparency, the query engine transforms queries to reflect the physical storage and indexing formats. Non-

existent views (i.e., partial cube materializations) are supported by way of surrogate views that are selected based upon size and dimension considerations. An optimized parallel sample sort supports re-ordering and aggregation features that ultimately return a result set consistent with the user's original specification.

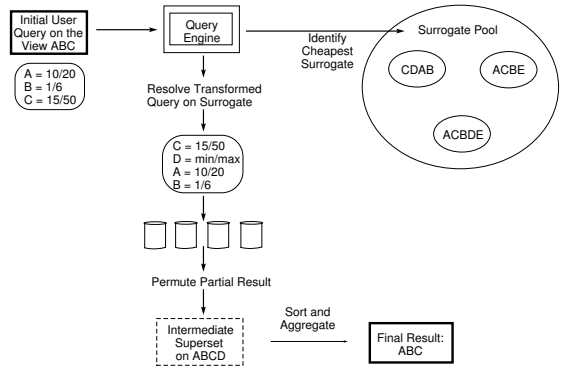

Figure 6: The process of resolving queries against materialized views. Note the view specified in the query may not exist in which case a surrogate will be selected.

In practice, OLAP queries tend to exploit dimension hierarchies such as time or product classification. Our server supports hierarchical query resolution by way of an extensive dimension-aware caching subsystem. Not only does the caching framework improve the response time for general queries, but it can be used to efficiently resolve common hierarchical OLAP queries. Specifically, roll-up, drilldown, slice and dice, and pivot queries can directly manipulate the intermediate hypercube results sets associated with previous hierarchical queries. We will show in our demonstration examples where arbitrary OLAP queries (including hierarchical dimensions) are submitted in batches of 1000 to 16-nodes of our Linux cluster and are answered at approximately 100 queries per second.

6 Demonstration Description

In our demonstration, we will show the efficiency of cgmOLAP's parallel data cube generation and querying algorithms. Our demonstration interface consists of three parts.

1. A cube specification panel that supports the selection of existing benchmark data sets or the specification of synthetic data sets in terms of number of rows, dimensionality, cardinality, skew, min support etc. From this panel, attendees can initiate the generation of full, partial, or iceberg cubes.
2. A query specification panel that allows the specification of parameters to our synthetic query generator or allows queries to entered directly by attendees. Our synthetic query generator produces batches of random range, rollup, drilldown, slice, dice and pivot queries to be evaluated against randomly selected views which may or may not have been materialized.
3. A performance monitoring panel which supports the visualization of cube generation and querying performance parameters such as rows generated per second and queries evaluated per second.

For this demonstration, the cgmOLAP system will be running on two 32 processor Linux-based clusters, one located in Canada and the other in Australia. In the unlikely event that both of these machine become unavailable over the Internet during the demo session, the cgmOLAP system will be run locally on a single processor machine using MPI's virtual processor simulation.

References

[1] The PANDA project. http://www.cs.dal.ca/∼panda/.

[2] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. *SIGMOD*, 1999.

[3] Y. Chen, F. Dehne, T. Eavis, and A. Rau-Chaplin. Parallel ROLAP data cube construction on shared-nothing multiprocessors. In *Int. Parallel and Distributed Processing Symposium*, 2003.

[4] Y. Chen, F. Dehne, T. Eavis, and A. Rau-Chaplin. PnP: Parallel and external memory iceberg cube computation. In *ICDE (to appear)*, 2005.

[5] Y. Chen, F.Dehne, T.Eavis, and A.Rau-Chaplin. Building large ROLAP data cubes in parallel. In *IDEAS '04*, 2004.

[6] F. Dehne, T. Eavis, and A. Rau-Chaplin. Parallelizing the datacube. *Distributed and Parallel Databases*, 11(2):181–201, 2002.

[7] F. Dehne, T. Eavis, and A. Rau-Chaplin. Computing partial data cubes. In *HICSS-37*, 2004.

[8] The Rising Storage Tide, 2003. www.datawarehousing.com/papers.

[9] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. *VLDB*, pp 299–310, 1998.

[10] S. Goil and A. Choudhary. High performance OLAP and data mining on parallel computers. *Journal of Data Mining and Knowledge Discovery*, (4), 1997.

[11] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[12] H.Lu, J.X. Yu, L. Feng, and X. Li. Fully dynamic partitioning: Handling data skew in parallel data cube computation. *Distributed and Parallel Databases*, 13:181–202, 2003.

[13] R. Ng, A. Wagner, and Y. Yin. Iceberg-cube computation with PC clusters. *SIGMOD*, pp 25–36, 2001.

[14] S. Sarawagi, R. Agrawal, and A. Gupta. On computing the data cube. Technical report rj10026, IBM Almaden Research Center, San Jose, CA, 1996.

[15] The Winter Report. www.wintercorp.com/vldb/.

[16] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. *VLDB*, 2003.