PnP: Parallel And External Memory Iceberg Cube Computation

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

Motivated by the work of Ng et.al. [2] and the recent success of another hybrid sequential method, Star-Cubing [4], we further investigate the use of hybrid approaches for the parallel computation of very large iceberg-cube queries. We present "Pipe 'n Prune" (PnP), a new hybrid method for iceberg-cube query computation. The novelty of our method is that it achieves a tight integration of top-down piping for data aggregation with bottom-up Apriori data pruning. 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 is a hybrid, sort-based, algorithm for the computation of very large iceberg-cube queries. The idea behind PnP is to fully integrate data aggregation via top-down piping [3] with bottom-up (BUC [1]) Apriori pruning. We introduce a new operator, called the *PnP operator*. 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 [3]) with iceberg-cube pruning. (2) It uses these prefix group-bys to perform bottom-up (BUC [1]) Apriori pruning for new group-bys that are starting points of other piping operations. An example of a 5-dimensional PnP operator is shown in Figure 1a. 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 1b.

We performed an extensive performance analysis of PnP for all of the above mentioned scenarios with the following main results: In the first scenario, PnP performs very well for both, dense *and* sparse data sets, providing an interesting alternative to BUC and Star-Cubing. In the second scenario, PnP shows a surprisingly efficient handling of disk I/O, with an external memory running time that is less than twice the running time for full in-memory computation of the same iceberg-cube query. In the third scenario, PnP scales very well. In [2], 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 16 processors. In more detail, our analysis of PnP showed the following.

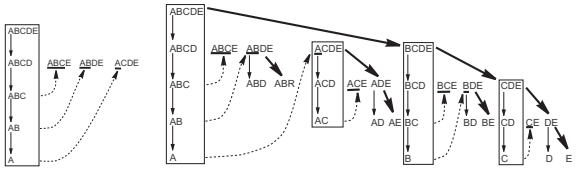

Figure 1. (a) A PnP Operator. (b) A PnP Tree. (Plain arrow: Top-Down Piping. Dashed Arrow: Bottom-up Pruning. Bold Arrow: Sorting.)

Sequential iceberg-cube queries: We performed an extensive performance analysis of PnP in comparison with BUC [1] and StarCube [4]. We observe that the sequential performance of PnP is very stable even for large variations of data density and data skew. Sequential PnP typically shows a performance between BUC and StarCube, while BUC and StarCube have ranges of data density and skew where BUC outperforms StarCube or vice versa (Figure 2a). For the special case of full cube computation, PnP outperforms both BUC and StarCube (Figure 2b).

External memory iceberg-cube queries: Since PnP is composed mainly of linear scans and does not require complex in-memory data structures, it is conceptually easy to implement as an external memory method for very large iceberg-cube queries. In order to make good use of PnP's properties, we have implemented our own I/O manager to have full control over latency hiding through overlapping of computation and disk I/O. Our experiments show minimum loss of efficiency when PnP switches from in-memory to external memory computation. The measured external memory running time (where PnP is forced to use external memory by limiting the available main memory) is less than twice the running time for full in-memory computa-

Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)
1084-4627/05 $20.00 © 2005 **IEEE**

tion of the same iceberg-cube query. In Figure 3 we observe similarly shaped curves even as we increase the dimensionality of the problem due in large part to the effects of iceberg pruning. The location of the slight jump in time, corresponding to the switch to external memory, occurs between 5 million rows and 7 million rows depending on the dimensionality of the iceberg cube being generated.

Parallel iceberg-cube queries on shared-nothing PC clusters (with multiple disks): PnP is well suited for shared-nothing parallelization, where processors do not share any memory and all data is partitioned and distributed over a set of disks. The full version of this paper presents a PnP parallelization which (1) minimizes communication overhead, (2) balances work load, and (3) makes full use of our I/O manager by overlapping *parallel* computation and *parallel* disk access on all available disks in the PC cluster. Extensive experiments show that our new parallel, external memory, PnP method provides close to linear speedup particularly on those data sets that are hard to handle for sequential methods. Most importantly, parallel PnP scales well and provides near linear speedup for larger numbers of processors (thereby solving the scalability problem observed in [2]). Figure 4 shows the running time and corresponding speedup for parallel PnP for input data sizes between $t = 1$ and 8 million rows. Near linear speedup is observed when there are at least $n/p = 500,000$ rows per processor. Figure 5 shows the running time and corresponding speedup for increasing dimensionality. Again we observe near optimal linear speedup all the way up to 16 processor. With 32 processors the parallel version of PnP achieves at least 50% speedup when generating cubes of between 8 and 10 dimensions and near optimal linear speedup when generating a 11 dimensional cube. Note that the best speedup is achieved on the problems which are hardest to solve sequentially, that is those that involve the largest problems in terms of input size and/or dimensionality.

References

[1] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. *Proceedings of the 1999 ACM SIGMOD Conference*, pages 359–370, 1999.

[2] R. Ng, A. Wagner, and Y. Yin. Iceberg-cube computation with PC clusters. *Proceedings of 2001 ACM SIGMOD Conference on Management of Data*, pages 25–36, 2001.

[3] S. Sarawagi, R. Agrawal, and A.Gupta. On computing the data cube. Technical Report RJ10026, IBM Almaden Research Center, San Jose, California, 1996.

[4] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. *in Proceedings Int. Conf. on Very Large Data Bases (VLDB'03)*, 2003.

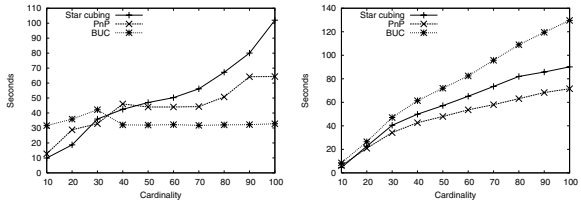

Figure 2. Sequential PnP. (a) Iceberg cube, varying cardinality. Fixed t=5M, d=6, s=0, m=100. (b) Full cube, varying cardinality. Fixed t=1M, d=6, s=0, m=1.

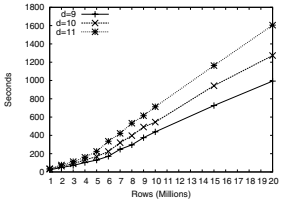

Figure 3. External Memory PnP. Varying dimensionality. Fixed c=300, m=1000, s=0, b=500M.

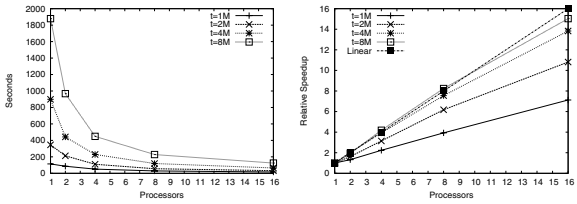

Figure 4. Parallel PnP. Varying input data size t. Fixed d=10, c=100, m=100, s=0. (a) Running Time. (b) Speedup.

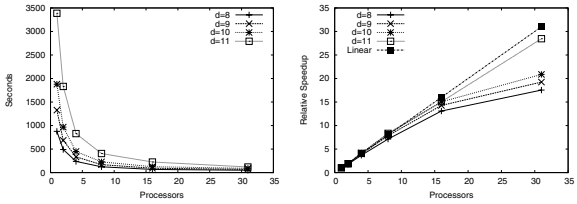

Figure 5. Parallel PnP. Varying dimensions. Fixed data size t=8M, c=100, m=100, s=0. (a) Running Time. (b) Speedup.