Brown Dwarf: A P2P Data-Warehousing System

Katerina Doka
katerina@cslab.ntua.gr

Dimitrios Tsoumakos
dtsouma@cslab.ntua.gr

Nectarios Koziris
nkoziris@cslab.ntua.gr

School of Electrical and Computer Engineering
National Technical University of Athens, Greece

ABSTRACT

In this demonstration we present the *Brown Dwarf*, a distributed system designed to efficiently store, query and update multidimensional data. Deployed on several commodity nodes, our system manages to distribute large volumes of data over network peers on-the-fly and process queries and updates on-line through cooperating nodes that hold parts of a materialized cube. Moreover, it adapts its resources according to demand and hardware failures and is cost-effective both over the required hardware and software components. All the aforementioned functionality will be tested for various datasets and query loads.

Categories and Subject Descriptors

H.3.4 [**INFORMATION STORAGE AND RETRIEVAL**]: Systems and Software—*Distributed systems*

General Terms

Design

Keywords

Data Warehousing, Data Cube, P2P

1. INTRODUCTION

Data warehousing has become a vital component of organizations and companies, which heavily rely on data analysis in order to identify behavioral patterns. Moreover, constant data analysis is needed to immediately detect real-time changes in trends. Yet, data warehouses present a strictly centralized and off-line approach in terms of data location and processing ([5, 6]). Even some works proposing distributed warehousing systems ([2, 1]) basically just interconnect a number of warehouses, maintaining their centralized functionality.

We have created an always-on, distributed data warehousing system, the *Brown Dwarf* (*BD*) [3], where geographically spanned users, without the use of any proprietary tool, can share and query information. Moreover, it employs a robust and efficient adaptive replication scheme, perceptive both to workload skew and node churn using only local load measurements and overlay knowledge.

2. SYSTEM DESIGN

The essence of *BD* is the distribution of a highly effective, centralized structure, the *Dwarf* [5], over the nodes of an unstructured P2P overlay on-the-fly. Each vertex of the dwarf graph (dwarf node) is designated with a unique ID and assigned to a network node. Adjacent dwarf nodes are stored in adjacent network nodes

CIKM'10, October 25–29, 2010, Toronto, Ontario, Canada.
ACM 978-1-4503-0099-5/10/10.

Table 1: A sample fact table

DIM1	DIM2	DIM3	Measure
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

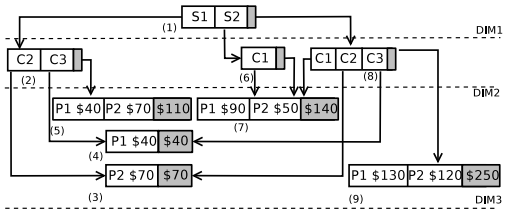

Figure 1: Centralized dwarf structure over the data of Table 1

in the P2P layer by adding overlay links, which represent the edges of the centralized structure. Each peer maintains a *hint* table, necessary to guide a query from one network node to another until the answer is reached. The *hint* table is of the form (*currAttr, child*), where *currAttr* is the attribute of the query to be resolved and *child* is the ID of the dwarf node it leads to. In case of a leaf node, *child* is the aggregate value.

Pictorially, Fig. 2 shows that nodes (1)–(9) are selected in this order to store the corresponding dwarf nodes of Fig. 1, forming an unstructured P2P overlay. Queries and updates are then handled using the same path that would be utilized in *Dwarf*, with overlay links now being followed: An incoming query about S1 will be forwarded to node (2). From there, depending on the requested group-by (ALL, C2 or C3), nodes (3), (4) or (5) can be visited.

Insertion The creation of the data cube is undertaken by a specific node (*creator*), that has access to the fact table. The creator follows the algorithm of the original dwarf construction, distributing the dwarf nodes on-the-fly during the tuple-by-tuple processing, instead of keeping them in secondary storage. The creation of a cell corresponds to the insertion of a value under *currAttr* and the creation of a dwarf node corresponds to the registration of a *child*.

Incremental Updates The procedure of incremental updates is similar to the insertion process, only now the longest common prefix between the new tuple and the existing ones must be discovered following overlay links. Once the last common attribute is discovered, underlying dwarf nodes are recursively updated. This means that dwarf nodes are expanded to accommodate new cells for new attribute values and that new ones are allocated when necessary.

Query Resolution Queries are resolved by following their path along the overlay attribute by attribute. Each query attribute belongs to a dwarf node which, through its hint table, leads to the network node responsible for the next one. Since adjacent dwarf nodes belong to overlay neighbors, the answer to any point or group-by

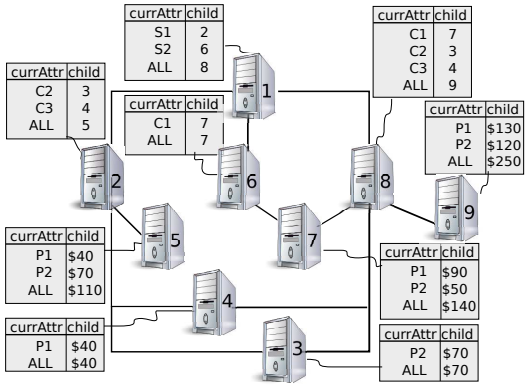

Figure 2: The dwarf distribution over the overlay nodes

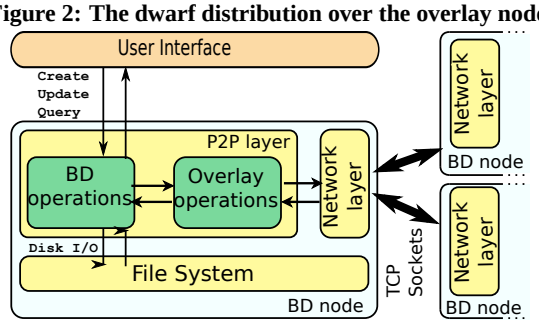

Figure 3: Architecture of a *BD* network node

query is discovered within at most d hops, where d is the number of dimensions. The existence of a single entry-point, which constitutes a single point of failure, is tackled by our replication strategy.

Adaptive Mirroring *BD* adopts a replication scheme adaptive to both node churn and data skew. Initially, a global replication parameter k defines the degree of data redundancy: During the insertion phase, k+1 instances (*mirrors*) of each dwarf node are being stored. Monitoring its load on a per dwarf node basis, a network node hosting an overloaded dwarf node can create additional mirrors through the *expansion* process. The newly created mirror will be used by the parent node(s) in order to receive some of the requests. In the opposite case, an underloaded dwarf node can be deleted from the system through the *shrink* process, as long as the total number of its mirrors remains over k+1. The combination of *expansion* and *shrink* enables *BD* to obtain increased resources to handle spikes in load and release them once the spike has subsided.

3. DEMONSTRATION SCENARIO

The BD system has been entirely developed in Java and deployed on an actual testbed of 16 LAN commodity nodes (dual core, 2.0 GHz, 2GB of main memory). Fig. 3 depicts the architecture of a system node. *BD* is accessible through a Java GUI that exposes its functionality to the user and allows her to perform insertions, updates and queries over the data cube. The P2P layer consists of the *Warehouse Operations* and the *Overlay Operations* components. The former is responsible for manipulating system-specific messages, orchestrating the mirroring process and interacting with the local filesystem in order to store and retrieve dwarf nodes. The latter handles the translation of system operations to overlay messages and backwards. These messages pass through the network layer, where they are transmitted or received through TCP sockets.

Creating a Cube The user will be able to choose from a series of datasets to create the corresponding distributed cube. The datasets will be of various sizes, dimensions and densities, both real and synthetically generated. Upon dataset choice, its characteristics are shown on screen. The user can also set the replication factor k and

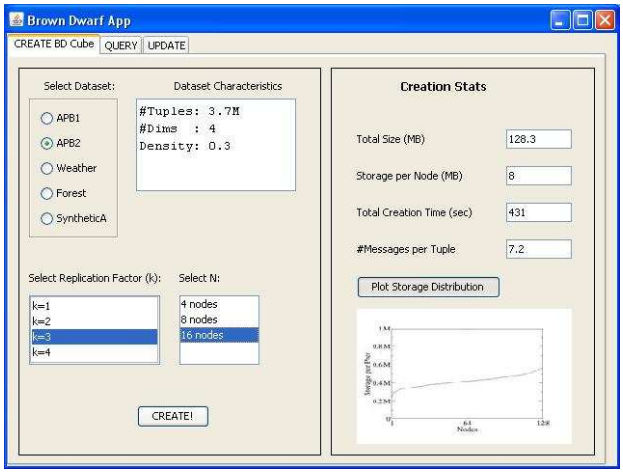

Figure 4: Screen capture from the cube creation tab

the number of network nodes that will participate in the P2P overlay. After the cube creation, important statistics and performance metrics will be presented: The creation cost in terms of time and network messages and the total storage consumed by the created cube. Finally, users will be able to have a graphical overview of the storage distribution per network node, as the corresponding graph can be displayed on demand. The initial GUI form for the creation process can be seen in Fig. 4.

Querying the Cube After cube creation and navigating to the QUERY tab, the user will be able to choose one of the predefined workloads in order to query the system. The available workloads will include various number of queries with various levels of skew. Similarly to the creation tab, their individual characteristics are displayed on screen. The rate at which queries will be sent to the system is user-defined. After pressing on the Send button, the workload is being processed and statistics are gathered. Besides confirming the 100% precision of our system, we will demonstrate all performance parameters for the processed query-load, namely response times and average load per node.

Updating the Cube This third part of the demonstration relates to applying incremental updates to the system on-line (UPDATE tab). Users will be given the chance to initiate updates one by one, or in bulk, by selecting one of the predefined update sets. As before, the application will present the appropriate performance metrics to the user, showcasing performance (in time elapsed versus the number of updates).

Performance Insight Our initial evaluation on an actual testbed of 16 LAN nodes has proven that Brown Dwarf manages to distribute the structure across the overlay nodes incurring only a small storage overhead compared to the centralized algorithm. Moreover, it accelerates cube creation up to 5 times and querying up to several tens of times by exploiting the capabilities of the available network nodes working in parallel. More details can be found in [3] and [4]

4. REFERENCES

[1] S. Abiteboul, T. Allard, P. Chatalic, et al. WebContent: Efficient P2P Warehousing of Web Data. *VLDB'08*.

[2] M. Akinde et al. Efficient OLAP Query Processing in Distributed Data Warehouses. *Information Systems*, 28(1-2):111–135, 2003.

[3] K. Doka, D. Tsoumakos, and N. Koziris. Brown Dwarf: Distributing the Power of OLAP. In *HPDC'10*.

[4] K. Doka, D. Tsoumakos, and N. Koziris. Brown dwarf: Distributing the Power of Olap to Unstructured P2P overlays. Technical report, NTUA, Dec 2009.

[5] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: Shrinking the PetaCube. In *SIGMOD'02*.

[6] W. Wang, H. Lu, J. Feng, and J. X. Yu. Condensed Cube: An Effective Approach to Reducing Data Cube Size. In *ICDE'02*.