Future Generation Computer Systems 26 (2010) 259–266

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Parallel OLAP with the Sidera server

Todd Eavis *, George Dimitrov, Ivan Dimitrov, David Cueva, Alex Lopez, Ahmad Taleb

Concordia University, Montreal, Canada

ARTICLE INFO

Article history:
Received 16 May 2008
Received in revised form
26 September 2008
Accepted 5 October 2008
Available online 25 October 2008

Keywords:
OLAP
Cluster computing
Parallel DBMS

ABSTRACT

Online Analytical Processing (OLAP) has become a primary component of today's pervasive Decision Support systems. As the underlying databases grow into the multi-terabyte range, however, single CPU OLAP servers are being stretched beyond their limits. In this paper, we present a comprehensive model for a fully parallelized OLAP server. Our multi-node platform actually consists of a series of largely independent sibling servers that are "glued" together with a lightweight MPI-based Parallel Service Interface (PSI). Physically, we target the commodity-oriented, "shared nothing" Linux cluster, an architecture that provides an extremely cost effective alternative to the "shared everything" commercial platforms often used in high-end database environments. Experimental results demonstrate both the viability and robustness of the design.

1. Introduction

Contemporary data warehouses (DWs) now represent some of the world's largest database systems, often stretching into the multi-terabyte range. Structurally, data warehouses are based upon a *denormalized* logical model known as the *Star Schema*. The "star" consists of a large *fact* table that houses the organization's measurement records, coupled with a series of smaller *dimension* tables defining specific business entities (e.g., customer, product, store). Given the enormous size of the fact tables, however, it can be extremely expensive to query the raw data directly. Typically, we augment the basic Star Schema with compact, pre-computed aggregates (called *group-bys* or *cuboids*) that can be queried much more efficiently at run-time. We refer to this collection of aggregates as the *data cube*. Specifically, for a d-dimensional space, $\{A_1, A_2, \ldots, A_d\}$, the cube defines the aggregation of the 2^d unique dimension combinations across one or more relevant *measure* attributes.

In practice, the generation and manipulation of the data cube is often performed by a dedicated OLAP server that runs on top of the underlying relational data warehouse. In other words, it is the job of the OLAP server (at least in part) to pre-process the warehouse data. The sheer scale of the DWs, however, has recently led researchers to explore opportunities for parallelizing or distributing the OLAP workload across multiple CPUs/disks. While "shared everything" parallel models are relatively attractive for small to medium sized warehouses, they tend to have limited scalability in terms of both the CPU count and the number of available disk heads.

In this paper, we describe an architecture for a scalable OLAP server known as Sidera that targets the commodity-based Linux cluster. The platform consists of a network-accessible frontend server and a series of protected backend servers that each handle a portion of the user request. A key feature of the cluster design is that each backend server requires little knowledge of its siblings. In effect, each node functions independently and merely interacts with a Parallel Service Interface (PSI) that, in turn, coordinates communication between nodes of the sibling network. The various constituent elements of the server have been evaluated experimentally and demonstrate a combination of performance and scalability that is particularly attractive given the use of commodity hardware and open source software.

The paper is organized as follows. In Section 2, we discuss related work. Section 3 presents a simple architectural overview, while Sections 4 and 5 describe the processing logic and system components for Sidera's frontend and backend servers, respectively. Experimental results are presented in Section 6, with Section 7 offering concluding remarks.

2. Related work

With respect to DBMS parallelism, early work focused on the exploitation of relatively exotic hardware in order to improve performance for transaction-based (OLTP) queries [1,2] By the 1990s, it had become clear that commodity-based "shared nothing" databases provided significant advantages over the earlier SMP architectures in terms of cost and scalability [3]. Subsequent research therefore focused on partitioning and replication models for the tables of the parallelized DBMS [4]. More recent research

* Corresponding author.
E-mail address: eavis@cs.concordia.ca (T. Eavis).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.10.007

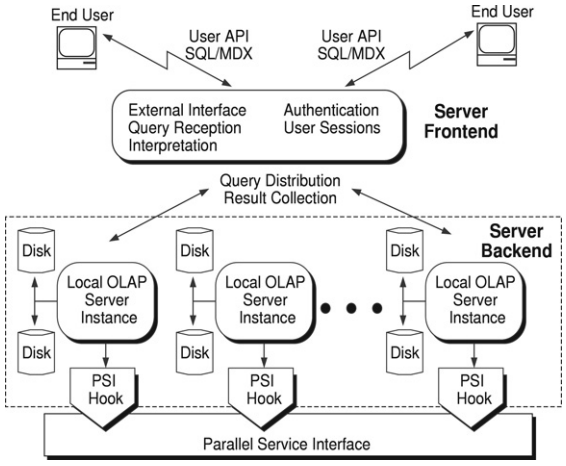

Fig. 1. The core architecture of the parallel Sidera OLAP server.

in parallel systems has generally been smaller in scope and has targeted issues such as static versus dynamic locking in parallel contexts [5], as well as parallel file systems for transparent RAID storage [6]. Finally, the Grid has also emerged as a novel target for distributed processing, and a number of papers have explored the design of protocols for scalable, Internet-based data management systems [25].

Commercially, of course, a number of vendors have pursued parallel implementations. Traditionally, such systems have targeted tightly coupled SMP hardware. More recently, vendors have begun to exploit more cost effective CPU and storage frameworks. IBM, for example, is actively exploring cluster implementations [7], while Oracle currently exploits a shared disk backend [8]. Such systems, however, are, by definition, proprietary and DBMS-specific.

In terms of OLAP/data cube research, a large number of papers have been published since Gray et al. delivered the seminal data cube paper in 1996 [9]. Initially, work focused on efficient algorithms for the computation of the exponential number of views in the complete data cube [10,11]. More recent methods have made an effort to support cubes that are both more expressive and more space efficient. The CURE cube, for example, supports the representation of dimension *hierarchies* and relatively compact table storage [12]. In fact, we note that OLAP-specific data structures, algorithms, indexing, and storage have consistently been shown to provide significant performance improvements relative to the facilities traditionally offered by relational database management systems. Stonebraker et al., for example, suggests that an order of magnitude improvement can be achieved via the design and optimization of domain specific DBMS systems (e.g., dedicated DW/OLAP servers) [24].

Finally, we have recently seen preliminary attempts to integrate parallelism and data warehousing/OLAP. On the one hand, a number of researchers have described methods for the parallelization of state-of-the-art sequential data cube generation methods [13,14]. On the other, fact table partitioning methods for cluster implementations have been explored. Perhaps the most interesting approach is one that proposes the *virtualization* of partial fragments over physically replicated tables [15]. We note, however, that in none of these cases do we see a comprehensive, "end-to-end" proposal for OLAP parallelism.

3. The Sidera architecture

Sidera has been designed from the ground up, to function as a parallel OLAP server. Node coordination is controlled by an MPI (Message Passing Interface) based Parallel Service Interface (PSI) that allows each node to participate in global sorting, merging, and aggregation operations. This ensures that the full computational capacity of the cluster is brought to bear on each and every query. Fig. 1 provides an illustration of the fundamental design. Note how the frontend node serves as an access point for user queries. Query reception and session management is performed at this point but the frontend does not participate in query resolution, other than to collect the final result from the backend instances and return it to the user. In turn, the backend nodes are fully responsible for storage, indexing, query planning, I/O, buffering, and meta data management. In addition, each node houses a PSI component that allows it to hook into the the global Parallel Service Interface.

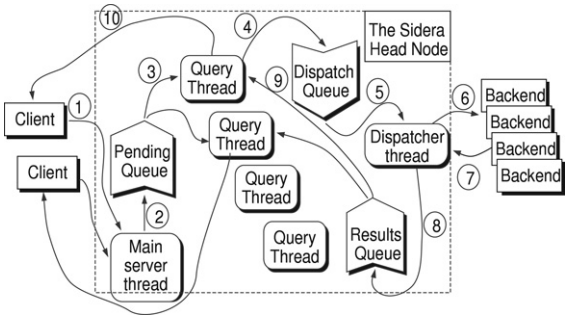

Fig. 2. Query processing cycle on the Sidera frontend.

In the remainder of the paper, we present a succinct overview of the frontend and backend server components. Though the focus is on the physical architecture, we will present algorithmic elements where necessary in order to understand the full functionality of the system.

4. The Sidera frontend

The Sidera frontend, or head node, represents the server's public interface. Its core function is to receive user requests and to pass them along to the backend nodes for resolution. Fig. 2 provides an illustration of the frontend architecture and processing logic. Here, one can see that processing is essentially driven by a sequence of three FIFO queues: the Pending Queue (PQ), the Dispatch Queue (DQ), and the Results Queue (RQ). Also, in keeping with current trends in server design, server functionality is based upon a lightweight, multi-threaded execution model. The threads themselves are based upon the POSIX *pthread* framework.

In short, as queries arrive, they are placed into the Pending Queue and subsequently retrieved by a set of persistent query threads (organized into a pre-initialized *thread pool*). Once a query thread is notified of a pending user query, it executes the steps outlined in Fig. 2. Upon successful authentication, a basic syntactical check of the query is performed to ensure that it should even be passed to the backend for resolution. If so, the query is deposited in the Dispatch Queue and a signal is sent to the sleeping Dispatch Thread. The Query Thread then goes to sleep. Eventually, the final query results will return to the frontend and the sleeping Query Thread will be notified that a result is waiting in the Result Queue. The final result is simply returned to the user, the client-specific socket is closed, and the Query Thread "returns" to the thread pool to await the next user request.

A key element of the frontend cycle is the work of the MPI Dispatcher thread. The job of the dispatcher is to interface with the backend query resolution nodes. Quite simply, once the thread is signaled that a new query is pending, it passes the query to each of the p nodes in the cluster and waits for the results to be deposited in a local buffer. Communication at this stage is implemented with standard MPI Broadcast and Gather operations. We note that, since the final result of the query request is globally sorted by the backend nodes (one of the functions of the Parallel Serve Interface)