CS 848: Paper Summaries
W Anthony Young - 20161423
September 22nd, 2004

David Dewitt and Jim Gray - Parallel Database Systems

Summary of Contents

This paper appears to present three major pieces of information: (1) The general trend to shared-nothing architecture, (2) the reasons why and methods by which data can be partitioned, and, (3) the effect partitioning has on the performance of operations.

The paper starts with a lengthy introduction discussing the current state of database hardware. The authors spend several pages discussing how and why custom hardware solutions have failed, and provide readers with a background for a discussion of systems built with modular components. Further in the introduction, the authors discuss the two types of parallelism: pipelined and partitioned. Pipelined parallelism refers to piping the output of one operation into the input of another. The two operations can then work together by processing in tandem. Partitioned parallelism refers to the splitting of input into pieces and running multiple copies of one operation on different processors. The operations can all run in tandem and hopefully complete the task faster.

Once the introduction is complete, the authors spend time treating the metrics used to measure the performance of a parallel database system: speedup and scaleup. Speedup refers to the ability of a larger system to complete a task faster than a smaller system. For example, a system twice as large should (ideally) complete a task in half the time as a smaller system. Scaleup refers to the ability of a larger system to complete a larger task in the same time that a smaller system completes a smaller task. For example, a system twice as large should (ideally) complete a task twice as large in the same amount of time as a task completed on a small system. We say ideally as the three barriers (startup, interference and skew) prevent total linear speedup and scaleup. Linear speedup would mean increasing the size of a system by N times would reduce the processing time of a task by N times. Linear scaleup would mean increasing the size of a system by N times would allow it to perform a task N times bigger in the same amount of time. A startup barrier is the amount of time it takes to start processes running. An interference barrier refers to processes needing to wait for each other to release resources, return results, etc. A skew barrier refers to the different amount of processing that needs to be done by each process. All three barriers affect performance.

After discussing metrics and barriers, the paper discusses the three different hardware organizations typically found in database systems along with their pros and cons. The three hardware systems are shared-memory (where each processor accesses a the same memory and hard drives), shared-disk (where each processor has its own RAM but accesses the same hard drives), and shared-nothing (where each processor has its own RAM and hard drives). The main disadvantages of shared-memory and shared-disk architectures are that they interfere with each other and are difficult to scale. This is because each processor must communicate with the common disks or memory, and scheduling can become a problem. This can be solved using affinities (i.e. physically relocating data used by each process to a separate disk) or through a very fast interconnection network between processors and disks. However, both these solutions

can be difficult and expensive to implement. Shared-nothing systems do not interfere with each other and can be scaled to hundreds and thousands of processors while still experiencing near-linear speedup and scaleup.

After discussing the pros and cons of the hardware architectures, the authors discuss some basics of the SQL language. This section is relatively straightforward with attention being paid to explanations of how operators work.

The authors then turn their attention to the methods of data partitioning and the reasons for partitioning data. Round robin, range and hash-based partitioning appear to be the three most common types of partitioning used on data. Again, this section is relatively straightforward. The major benefit of partitioning is its ability to exploit the I/O hardware during a parallel operation to increase the speed at which data is retrieved. The type of queries typically performed on a data set determine, in a loose sense, what partitioning style is typically used. Sequential reads warrant round robin style partitioning as each successive tuple can be read from a different disk. Hash-based partitioning works best on associative searches as all the data can be read from one disk instead of starting and running searches at each disk. Range partitioning allows sequential reads of ordered data to progress much faster as records can be returned in sorted order. The authors outline the pros and cons of each partitioning method.

From a discussion of partitioning comes a discussion of how partitioning can speed up the execution of operations. For example, range partitioning can greatly increase the speed of the sort-merge join. As well, hash-based partitioning can greatly increase the speed of the hash join.

As the paper draws to a close considerable attention is paid to the hardware architecture and partitioning methods employed by some of the current (1992) systems available. The authors then proceed to discuss how Grosch's law has been broken through the use of shared-nothing computers. Since they make use of off-the-shelf components, no economies of scale are experienced as we see with mainframes (where more expensive computers perform better). Shared-nothing computers break this by significantly reducing the cost-performance ratio. The paper concludes with a discussion of some current (1992) areas of research in parallel database systems.

Comments

In general, the paper is well written and full of useful information. However, it is slightly disorganized. The authors flip from discussion of one topic to another and back again. For example, the authors begin by discussing system architectures and the failure of mainframes. They then interrupt for a discussion of parallelism and return later to the idea of different computer architectures. This jumping makes the paper hard to follow.

The authors do not provide any information to back up claims such as "These shared-nothing architectures achieve near-linear speedups and scaleups…" (pp. 88, para. 8) and "Sort-merge join works well in parallel dataflow environments…" (pp. 93, para. 9). As well, several other claims are simply stated as fact and no explanation of how or why a claim is true is given. This happens in sentences such as "…but reaps none of the hardware interconnect benefits". What benefits? How are those benefits not reaped?

In conclusion, I was impressed with the discussions and information in the paper. I believe some more attention should have been paid to organization as well as backing up claims that were made.

Goetz Graefe - Encapsulation of Parallelism

Summary of Contents

This paper begins with a detailed description of the Volcano query processing system. The paper discusses the construction of the extensible system and the modules that combine to form its major feature set. The system is composed of approximately two-dozen modules and 15 000 lines of C code. The introduction then provides a short description of how Volcano's method of parallelization is different from other systems.

The authors continue their discussion of Volcano by describing the previous work that has led to some of the design decisions of the system. The discussion then turns to one of the two major models of parallelism that the authors present: the bracket model of parallelism. In the bracket model, a process template is used to run operators. The template provides input to and receives output from the operator. An external scheduler that must have some knowledge of the locus of control for each operator must schedule the templates. Thus, the bracket model is not extensible without additional code to create and schedule the locus of control.

After the discussion of the bracket model, the authors turn to the design decisions of the Volcano system itself. Each operator in Volcano is implemented as an iterator. This means that the operations support the open-next-close paradigm, and have some support functions built into them to provide the actual operations on data. State records are used to hold input and output arguments, as well as support functions. State records are then linked together using input pointers. The input pointers provide a means to traverse the query plan. Volcano also makes use of "anonymous inputs" that allow an operator to perform its job without knowledge of who provided the input data or who will use the output data. When open is called on an operator, it recursively calls the open command on all operators below it in the query tree. Records are then created and pinned to buffer pools before being sent to the requesting operator. The requesting operator can use, remove (unpin), or pass on a record to the next operator. Volcano makes use of virtual device pages in buffers to provide temporary storage for intermediate results. For example, records that are created by a join can be stored in this memory before being pinned to the buffer. This allows globally unique record id's to be maintained. Volcano uses demand-driven dataflow in which records are only returned when an operator requests them.

After discussing the Volcano system, the authors turn their attention to the second model of parallelism: the operator model of parallelism. The operator model implements parallelism through the use of an operator called exchange. The exchange operator is an iterator that supports interprocess communication through the use of ports. In this manner, an operator running on one processor can still provide data to an operator on a second processor. This is novel as most previous methods of parallelism required scheduling to be done by another process, adding much more overhead to the system. With the operator model, the exchange operator can simply be inserted into several places in the query tree and the query tree will be executed in parallel on other processors. The exchange operator supports both vertical parallelism (pipelining between operators) and

horizontal parallelism (distributing an operator over multiple processors). Vertical parallelism is supported through the direct use of the exchange operator's ports for interprocess communication between one producer and one consumer running on different processors. Horizontal parallelism is supported by forking producer and, if necessary, consumer processes. This means, for example, that data can be distributed over multiple disks and piped to several operators each performing a hash join. Once the processes are forked, they synchronize to share the port numbers they are all sending and receiving data on. Then, the producer portions of the operator can pass data to all the consumer processes at once. Similarly, consumer portions of the operator can receive from all producers. Thus, distributed data can be accessed in parallel.

After a lengthy example, the authors turn their attention to some small variations that were made to the operator to provide better functionality while sorting. This includes methods to keep data separated by producer and to merge data from producers.

The authors then turn their attention to the performance improvements made with the exchange operator. It appears that the exchange operator provides a significant speed boost over traditional schedule-based parallelism. The authors claim that setting packet sizes higher can lead negligible overhead from the exchange operator. How this is possible is not clear to the reader.

Once this discussion of performance is complete, the authors mention some future directions for their work as well as some summary points for the current work.

Comments

In general, I found this paper very interesting to read. The first time I read it, I found it difficult to follow. Numerous spelling and grammar mistakes changed the meaning of sentences and confused me. As well, the persistent misuse of commas made it difficult to separate concepts and run-on sentences. However, the main concepts presented by the paper were easier to understand after a second read. However, there are several small issues that must be discussed.

The authors use some terms, such as "mechanism" and "policy" (pp. 2, para. 4) that are never defined or given context. This makes it difficult to understand their relevance or importance. Also, pp. 4, para. 4 is so poorly worded that after reading it seven times, I will do not understand the concept being presented.

I thought the author's discussion of horizontal parallelism could have been clearer. The discussion is rushed and difficult to understand. As well, the example presented is difficult to follow because adequate background (such as a query tree or SQL statement) has not been provided. This makes it very difficult to follow without very careful scrutiny. My understanding is that examples should be easy to follow: this one is not.

Section 4.4 discusses variants of the exchange operator. However, I do not understand many of the concepts presented in this section, as they are not discussed very clearly. But, despite these few shortcomings, I still believe that the main ideas presented were presented well and are quite novel.