Production Database Systems: Making Them Easy is Hard Work

David Campbell

Microsoft Corporation
One Microsoft Way
Redmond, WA
USA
davidc@microsoft.com

Abstract

Enterprise capable database products have evolved into incredibly complex systems, some of which present hundreds of configuration parameters to the system administrator. So, while the processing and storage costs for maintaining large volumes of data have plummeted, the human costs associated with maintaining the data have continued to rise. In this presentation, we discuss the framework and approach used by the team who took Microsoft SQL Server from a state where it had several hundred configuration parameters to a system that can configure itself and respond to changes in workload and environment with little human intervention.

1. Introduction

In order to optimally service requests, database systems must manage available resources such as memory and processing power; understand the distribution of the stored data to make good choices about querying the data; and balance the needs of competing requests into the system. The first generation of database systems forced administrators to make many of these choices, up front, before starting the database through a set of configuration parameters or a configuration file. Changing these parameters typically required that the database system be stopped and then restarted to adopt the new configuration profile. Thus, reconfiguration was very costly in a highly available environment. Furthermore, the skill set required to truly comprehend the meaning of hundreds of configuration parameters and the interaction between them was very high. Many systems in the field performed poorly due to mis-configuration.

In the mid-1990's the complexity of these systems was beginning to exceed the capacity of the existing talent pool to manage them. Commercial database vendors began to recognize that future wide scale adoption of database technology would require more efficient ways to manage these systems. In response to this, several products began introducing features to codify the knowledge required to "set the knobs". These "configurators" let less skilled users configure the system based upon a series of questions or rudimentary workload analysis. Adopting the recommendations of these configurators still required changes to the underlying configuration set and, typically, a restart of the system.

The team that architected Microsoft SQL Server 7.0 took a radically different approach to this problem. Rather than adding features to help "turn the knobs", they took a holistic approach that focused on eliminating the knobs while simultaneously maintaining administrative control where necessary. This approach focused on three major themes:

- Closed loop control
- Intention based design
- Profile guided configuration and tuning

2. Closed loop control

The Microsoft SQL Server team used control theory technologies long known in other engineering disciplines to encode closed loop, feedback based, systems for configuring many elements of the system in near real-time. These technologies are used in SQL Server to control the size of the buffer pool, growing and shrinking

Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004

the size of the overall memory pool dynamically in response to memory pressure from other processes on the running system. By default, SQL Server is configured to completely control the amount of memory it consumes, however, an administrator can define upper and lower bounds on the amount of memory used and the control algorithm will be constrained to honor those boundaries. This is an example where, by default, the system manages itself, however administrative policy can be enforced if desired. These dynamic control algorithms are also used elsewhere in the system such as automatically configuring the read ahead and write behind depth for prefetch and bulk write operations.

Not only have these closed loop algorithms eliminated a large number of configuration parameters, they also have the advantage of responding to external forces to automatically reconfigure the running system in response to changing conditions resulting in more efficient use of system resources. For example, in many database systems, an administrator must set aside a portion of system memory for various needs such as working memory for sorting, or storing compiled SQL plans. In SQL Server, the system dynamically manages these different memory needs based upon system conditions – only allocating sort working memory when there is a sort operation being performed. Thus, memory is a fungible resource that can be employed wherever it can do the most good at any instant in time.

3. Intention based design

The control surface of most database products evolved as the implementers added new configuration parameters as they added features. As a result, system administrators were forced to wrestle the existing set of knobs into the right form to do their jobs. So, while an administrator might want to ensure that restart recovery completed in less than 60 seconds, he may have to set checkpoint frequency, number of outstanding dirty log blocks and several other parameters to achieve the desired state. Intention based design turns this around by aligning the control inputs of the database product with the specific objectives of the administrator. So, the administrator can specify they want restart recovery to complete in 60 seconds rather than manipulating a number of other controls. Obviously, there may be conflicts between the specified inputs; with intention based design however, it is up to the database product to understand and negotiate these constraints rather than the administrator.

4. Profile guided configuration and tuning

During the process of trying to understand the costs of mis-configured systems, the Microsoft SQL Server team realized that there were a number of administrative tasks such a maintaining table and index statistics or providing an optimal index set for a particular workload that could be automated. As a result, the product development and research teams created features to automatically create and maintain distribution statistics on stored data and to recommend a set of optimal indexes for a specified workload.

Automatic statistics update can create index and column level distribution statistics to improve the cost based query optimization decisions. Since this is done automatically in response to knowledge required to perform effective query optimization, the system actually "learns" as it processes new queries. If a query plan choice could benefit from column level statistics, the system will schedule building the statistics so subsequent query activity can benefit from the knowledge.

The Index Tuning Wizard [1] can process a previously recorded workload and, in cooperation with the query optimizer, propose an optimal set of indexes to process the workload balancing both the query benefit from the indexes and the costs required to maintain the index set. Ultimately, the goal is to eliminate the need for the CREATE INDEX statement and to have the system maintain optimal indexing based upon system needs.

5. Conclusion

A holistic approach to "Ease of Use", coupled with use of well known techniques from other engineering disciplines can be used to build very sophisticated database systems where the intent of the administrator, coupled with workload analysis, can be used to dynamically control the configuration of the system. Use of closed loop control, intention based design, and profile guided tuning can result in a system that is more responsive; resource efficient; and much less prone to mis-configuration Systems build with these techniques require much less direct human input to maintain; freeing administrators to perform tasks that provide much more direct business value.

References

[1] Chaudhuri, S., Narasayya V., "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server." Proceedings of the 23rd VLDB Conference Athens, Greece, 1997, pages 146-155