Adabas

Technical Article

Adabas Parallel Services

By Rainer Herrmann
Techical Marketing Manager
Software AG
June 2007

Table of Contents

History

Adabas is known for its outstanding performance, in particular, its ability to process large data volumes without performance degradation, its minimal overhead with respect to system administration, and its high level of stability and reliability (continuous operation, efficient and reliable error recovery).

Traditionally, Adabas used multi-threading to service many users concurrently. For many sites this was a good compromise between throughput requirements and keeping processing overhead, which was needed to guarantee data integrity, to a minimum. However, there were many sites and applications which required even more throughput than multi-threading offered. Software AG then introduced Adabas SMP which made use of multi-processing. Adabas SMP minimized the overhead for data integrity by way of one update nucleus and several read-only nuclei.

When IBM offered its cluster solution for mainframes, Parallel Sysplex, Software AG developed a “data-sharing” solution, Cluster Services, to support the capabilities of Parallel Sysplex. With Cluster Services, several nuclei could run in a single operating system image (LPAR) and on different CPUs in a Parallel Sysplex environment. The aim was to make the combined processing power of several CPUs available for a single database and to increase availability and continuous operation further by avoiding single points of failure. Gone was the single update nucleus as an instance for a single point of failure. Now all nuclei were homogenous and capable of update processing. In case of failure of a nucleus, a peer to peer level recovery mechanism was established so that such a failure would not be noticeable by application users of the database. The system would remain up and running.

To keep the overhead for Cluster Services within limits, a number of innovative new design concepts, like an optimistic lock protocol for internal locking of RABN, were introduced and implemented in Cluster Services. The development effort was substantial and probably the biggest project in the history of Adabas development.

software AG

The success of Cluster Services convinced Software AG that the same principles could be applied for sites, which required more throughput but within a single operating system image.
Adabas SMP was superseded by a new product called Parallel Services. The objectives were better throughput, less CPU overhead and a more robust and reliable design. The same peer to peer recovery mechanism would apply to increase availability and continuous operation.

Objectives of Parallel Services

The main development objectives for Adabas Parallel Services were:

- Improved performance:
  The throughput (Adabas commands per time unit) was to be increased by using CPU resources from several processors concurrently. The throughput should be scalable, that is, it should increase as linearly as possible with the number of nuclei in a cluster. When compared with using a single Adabas nucleus, Parallel Services will result in a small but unavoidable increase in CPU consumption. This CPU consumption, however, should remain within narrow limits, and should increase only very slightly relative to workload whenever the number of nuclei is enlarged.

- Application transparency:
  The behavior of Adabas Parallel Services towards application programs must be compatible to that of a single Adabas nucleus. For an application program, it should be transparent whether it is running with a single nucleus or with Adabas Parallel Services, and it should also be transparent as to which specific nucleus within a cluster is processing the application program's commands.

- Increased availability:
  If an Adabas nucleus aborts, the other nuclei can continue to run practically interruption-free. (No single point of failure in our software).

Parallel Services has less CPU overhead than Adabas Cluster Services. The main reason for this is that a typical operation against the coupling facility costs approximately one fourth to one third of the average CPU time required by an Adabas command in a well tuned OLTP environment. In Parallel Services the function of the coupling facility is substituted by operations in a data space. These operations are cheaper since, compared to the coupling facility, all communications remain within the same operating system. Communications between different CPUs, which are responsible for most of the CPU overhead in Cluster Services, do not exist in Parallel Services.

Architecture

With Adabas, one nucleus works on one database. With Parallel Services, up to 31 nuclei can work on the same database. Each cluster nucleus runs its own process in its own address space connected to a data space. The datasets which comprise the database are accessible from all nuclei.
For synchronization, the nuclei use a common cache structure and a lock structure in the data space. In addition, the nuclei exchange messages by sending internal Adabas calls to each other.
A single Adabas nucleus uses a Work dataset for storing intermediate results as well as *protection data* which is required for *restart recovery*. In addition, the nucleus writes protection data to two Protection Log datasets which are used for *archive recovery*. In Parallel Services, each nucleus has its own Work and Protection Log datasets, which are written to only by that nucleus, but which can be read by all other nuclei.
The functions of the nucleus in Adabas Parallel Services are fully symmetrical: each nucleus can execute all operations. However, there are operations which at a given point in time can be executed by only one nucleus. An example of such an operation is a bufferflush, which writes changed blocks from the cache structure to the database. Still, the next bufferflush can be executed by a different nucleus.

Database Block Operations

Adabas Parallel Services uses a cache structure in the data space. However, whenever virtual 64-bit allocation is available the cache space can be allocated via an ADARUN Parameter in a 64- bit cache space outside of the data space. The cluster nuclei write all database blocks, which they have changed, to this cache structure. The bufferflush process writes the changed blocks regularly from the data space to the database. Blocks which have not been changed by the nuclei are still registered with the cache structure, regarding the presence of copies of these blocks in the local buffer pool of the individual nuclei, so that these copies can be marked as invalid whenever any of these nuclei write changed versions of these blocks to the global cache.

Block Updates without Locking

Adabas Parallel Services uses an optimistic method for reading and changing blocks, whereby the blocks are not locked. A compare and swap logic is used to write changed blocks to the cache structure. This function performs the write operation only if no other nucleus has written the block to the cache after the time at which the block was last read or written by the first nucleus. If the write operation fails because of a conflicting change, the nucleus attempting the write operation must re-read the changed block, repeat its own changes, and again request the write operation to the cache.

This method of allowing conflicting changes to blocks is possible because locks at the data record level prevent different cluster nuclei from updating simultaneously the same object in a block. In that conflicting updates always apply to different parts of a block, they can be repeated if the underlying block is changed abruptly.

Most index update operations involve only a single block at the lowest level (Normal Index) of the index tree. In this case, Adabas Parallel Services changes the block without previously locking it. If the write operation to the cache for the changed block fails because another nucleus has changed the same block, then the first nucleus must re-read the block and repeat its update. It is possible that in the interim the index structure has been changed. In this case, the nucleus must also repeat the positioning within the index.

Some update operations change the index structure. That is, they change fields in the upper index levels which contain physical references to lower levels. A typical example is an update which splits one index block into two because it ran full. A reference must then be entered at the next highest level to indicate the location of the new block. This structure information is vital for index positioning as well as for read, search and update commands. Temporary inconsistencies can influence the correctness of parallel operations. Therefore, for complex updates which result in structure changes in the index, the blocks involved are locked entirely to prevent parallel usage during the update.

For a single Adabas nucleus, the synchronization of parallel index operations is performed by setting read and write locks in the control blocks which are used to manage the buffer pool. Because all participating operations have direct access to these control blocks, an efficient synchronization is possible.

In Adabas Parallel Services, the direct method of synchronizing conflicting index operations requires the usage of read and write locks in the data space.

Recovery

Although to a lesser extent than database operations, the Adabas recovery logic was also affected by the extension for Parallel Services. This involved not so much the backing out of individual transactions (transaction recovery) but rather the handling of error situations which cause an abort of an Adabas nucleus, as well as errors which cause damage to or destruction of the database.

Transaction and Restart Recovery

During operation, Adabas writes protection records for updates and transactions to the so called Work dataset. If the Adabas nucleus should abort, this protection information is used to return the database to a physically and logically consistent state.

The writing of changed blocks to the database occurs asynchronously to the execution of update commands and the changing of blocks in the buffer pool. Changed blocks can be written to the database before the updates have been confirmed by the end of the corresponding transaction (steal). However, they need not necessarily be written to the database at the end of the transaction (noforce). Therefore, during a *restart recovery* following an abort of the Adabas nucleus, all transactions which have ended but whose updates were not stored in the database prior to the abort, must be repeated (*redo*, consequence of noforce). Conversely, all updates, which were stored in the database as part of a transaction, which was not yet completed at the time of the abort, must be backed out of the database (*undo*, consequence of steal). The protection records, which the nucleus writes to the Work dataset, contain the necessary information for the redo and undo operations.
In Adabas Parallel Services, each nucleus has its own Work dataset, and only the nucleus itself writes protection data to its own Work dataset. However, each nucleus is able to read the protection records created by another nucleus within the cluster. The backout of individual transactions (for example, backout (rollback) commands) functions in a Parallel nucleus in the same way as with a single nucleus. The nucleus reads the required protection records from its own Work dataset and backs out the updates of the transaction.
If all active nuclei abort, the nucleus which is next started initiates the restart recovery. This is the *offline recovery*, during which the nucleus reads the protection records from the Work data sets of all previously active cluster nuclei, and performs in essence the same recovery logic as for a single nucleus. Before they can be applied, the protection records from the various Work datasets must be placed in chronological order. This is possible since each protection record contains a time stamp indicating exactly when it was created.
If one or more, but not all, nuclei should abort, the remaining nuclei perform the *online recovery*. Each nucleus allows a certain time for all currently running transactions to reach normal completion, following which all remaining activities are interrupted and the connections to the cache and lock structure are terminated. This results in the loss of all data in these structures. The nucleus then performs the offline recovery logic, after which all nuclei can connect to new cache and lock structures, and normal operations can continue. During this entire process, the user contexts remain intact in the nuclei which did not abort.

Conclusion

Customer tests have shown that the above described procedures for the improvement of Parallel Services have achieved impressive results. In a joint effort with IBM last year, more than 300,000 Adabas calls were executed per elapsed time second on a commercial IBM mainframe in IBM's Poughkeepsie, New York test center. This is 100% more than could have been accomplished with a single nucleus. This makes Parallel Services the ideal product for any site having to deal with high throughput requirements or which needs to prepare for sudden bursts of high activity.
The test was run on z/OS V1.8, which supports more than 128 GB of real memory in a single image. The CPU consisted of a z9 series with 32 processors supporting 64bit virtual memory.

Software AG
Corporate Headquarters
Uhlandstraße 12.
D-64297
Darmstadt/Germany
T: +49-61 51-92-0
www.softwareag.com