THE POSSIBILITIES ARE INFINITE FUJITSU

WHITE PAPER

BS2000/OSD
DAB – Disk Access Buffer
Intelligent Caching with AutoDAB

Issue June 2009

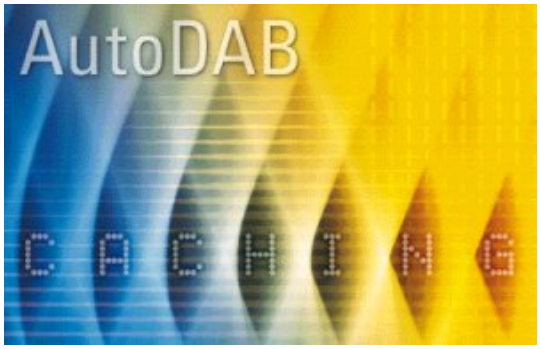
Pages 7

To cache or not to cache? That is not the question!

Business-critical computing is typified by high performance requirements and unpredictable load peaks in the underlying I&C infrastructures.

Smart workload management and data caching according to access profiles observed in the past are proven means of satisfying such high performance requirements and delivering the required service levels.

Data caching is employed today in all elements of the I&C infrastructure chain, i.e. in networks, in servers and in data storage.



Contents

Caching

A frequently used input/output caching method is caching in controllers in disk subsystems. In such cases the data transfer time is no longer dominated by the relatively slow disk access time, but by the time to access the disk controller‘s semiconductor memory. I/O transfers then run at data rates determined by the server I/O system and its channel structure.

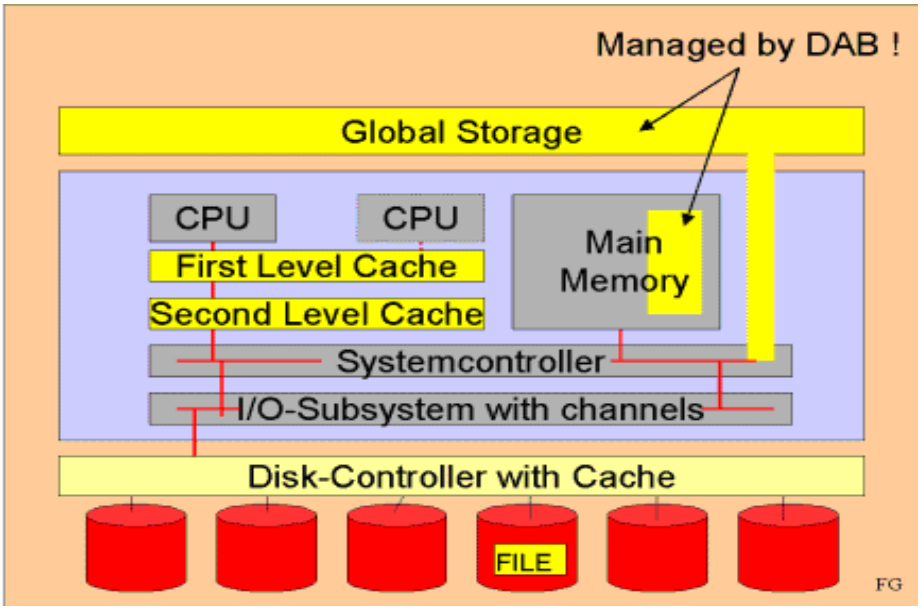
But highly advanced server architectures take this several steps further by “caching” data in even faster CPU-oriented semiconductor memories, e.g. in main memory, in global storage or in first-/second-level caches. Accesses by processors to these caches are an order of magnitude faster again, firstly because extremely fast semiconductor devices are used, and secondly because the I/O bottleneck with its I/O protocol overhead is bypassed and most access takes place synchronously.

BS2000/OSD and its servers provide all these caching mechanisms in a caching hierarchy.

Caching in main memories and global storages is handled by the BS2000 cache manager DAB. DAB implements CPU-oriented software caching, i.e. the CPUs synchronously access the cache memories “main memory” and/or “global storage”.

Data transfer is many times faster than controller caching in the disk subsystem. DAB configures the cache areas in the cache media, monitors caching performance, provides information services for status queries and handles inputs/outputs to the cache media.

As a BS2000 subsystem, DAB is, of course, easy to install, flexible and easy to use, and provides extensive automation options as well as allowing user-specific optimizations.



Servers cache with finesse

Servers read and store data and instructions with the aid of a storage hierarchy.

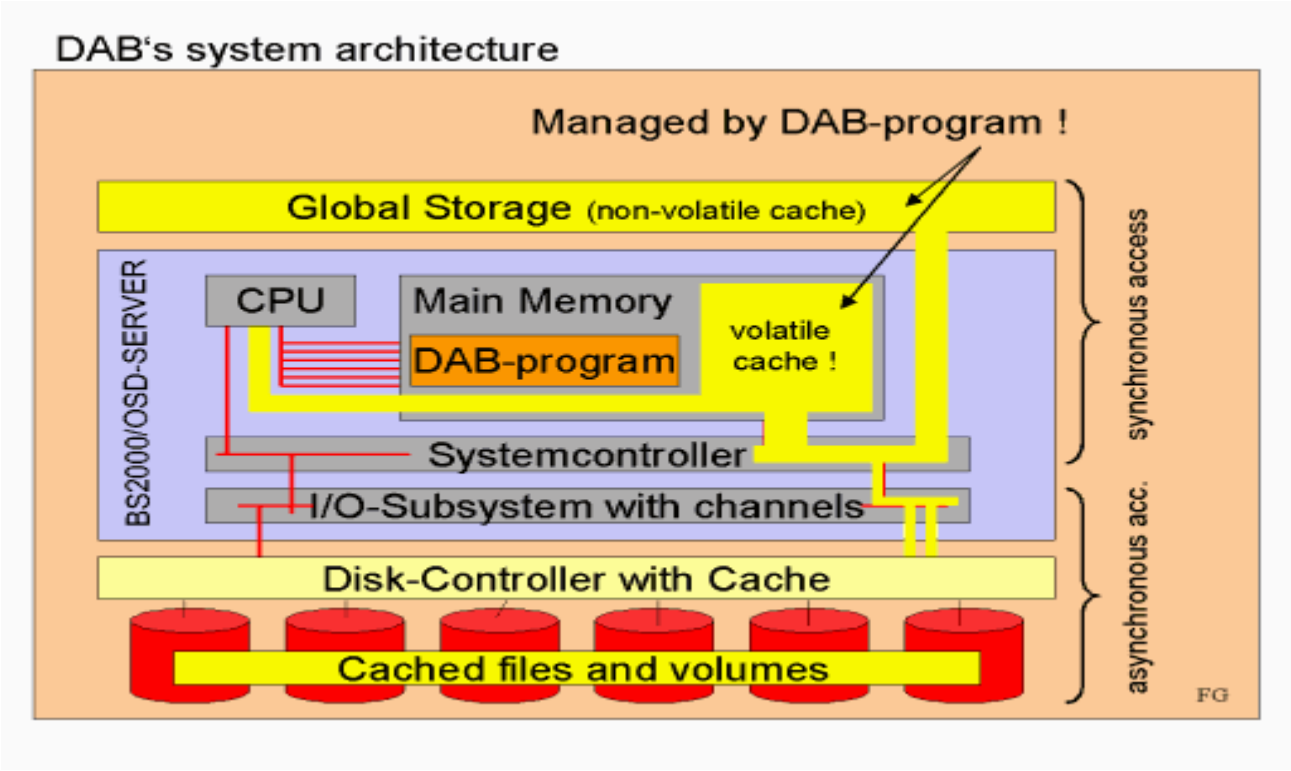
When a program starts, data and instructions are successively moved from relatively slow disk storage into main memory, from where they are fetched by the CPU. The main memory acts as a cache for the disks. From the CPU viewpoint that is still too slow, which explains why today's CPUs always have an L1 (Level 1) cache and typically also an L2 (Level 2) cache. When a CPU wants to execute a new instruction, it looks in the registers first. If the data is not there, it looks in the L1 cache, then in the L2 cache, and then in main memory. If the data is not there either, an I/O operation involving disk access is performed. From the CPU viewpoint, many cycles have been uselessly wasted in the meantime. The CPU gives up and turns to the next process. This context switching again costs in terms of numerous unproductive cycles.

This is where DAB caching enters the frame in BS2000/OSD servers. It introduces additional cache levels for synchronous CPU access.

Caching is everywhere

When you sit down at your desk in the morning, you expect instant access to frequently used information and resources. For example, telephone, writing materials and files relating to ongoing projects are exactly as they were left the previous day and so are immediately to hand on your desk. The PC is running, keyboard and mouse are active and you read the e-mail you have received since yesterday without any delay. Only in rare cases do you need to go to a filing cabinet to fetch less frequently used documents.

What lies behind this is the cache principle; in other words, the attempt to maintain the fastest possible access to frequently needed resources.



- DAB is the central cache manager in BS2000 for the CPU-oriented storage media main memory (read) and global storage (read and write).
- DAB sets up I/O cache areas in these storage media and implements the algorithms for read and write caches.
- DAB provides information services via the BS2000 interface and interworks closely with other BS2000 components.
- DAB caching can be used as an individually controlled mechanism or as an automated (AutoDAB) procedure.
- DAB can also work with encrypted files!

Advantages and benefits of DAB – an example

Advantages compared to controller caching in the disk subsystem

- Synchronous high-speed access by CPUs to DAB storage media
- Totally parallel IO cache access, i.e. no disk-specific serialization
- Automatic selection of the data to be cached at file level
- Dynamic cache ↔ database assignment, according to data flow behavior
- DAB is independent of the disk storage hardware
- DAB media size is configurable and can be changed at any time.

An example of DAB high-speed access (optimum case)

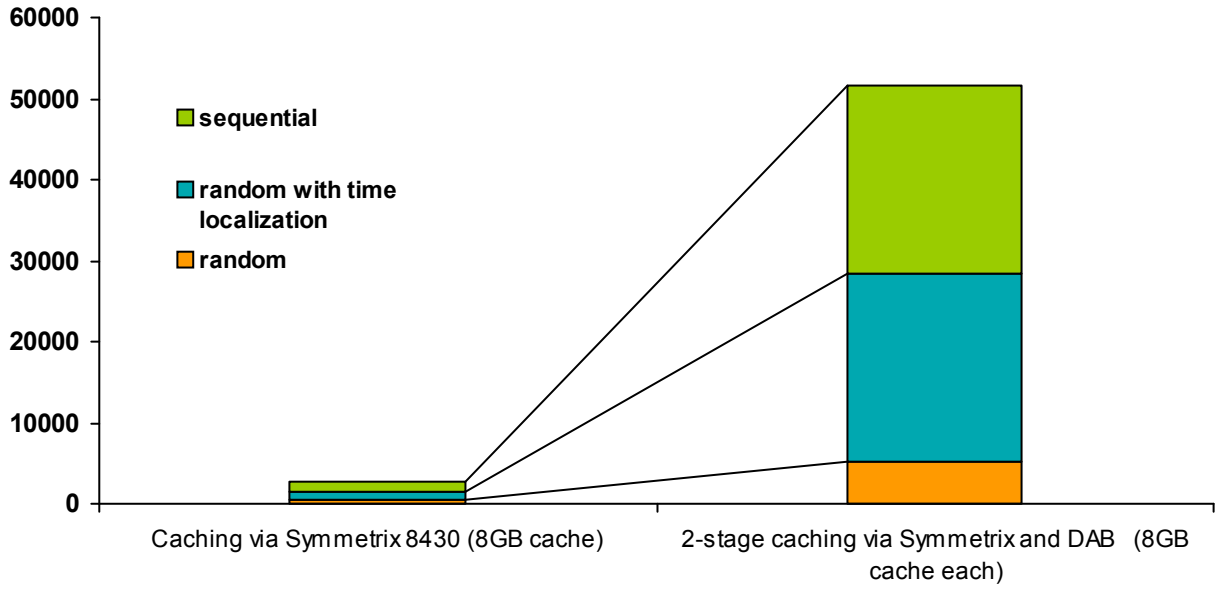
Performance test results achieved with the “AutoDAB” option:

System configuration

Server: S170-40

Disk subsystem: Symmetrix 8430, 8 GB cache, two 5-disk pubsets used

Global storage: 8 GB, used by one cache area



The chart shows how data throughput is increased as a result of attaching a DAB cache area and using AutoDAB. Compared with processing via the Symmetrix cache, the number of data accesses/sec increases from 2700 to 51,500. ***This is equivalent to a 1900% increase!***

How is this incredible increase achieved?

The outstanding hit rate in the cache is produced by the following factors:

- The **sequential files** have a very high hit rate because of a very large prefetch (data areas are stored in the cache in advance) and are then kept practically resident in the DAB cache because of the data volume.
- The **random files with time localization** are likewise served with very high hit rates thanks to AutoDAB’s intelligent algorithms and occupy only a small percentage of cache memory.
- Intelligent use of cache memory by DAB means that a large proportion remains available for caching files with **random** access patterns, which thus also helps to boost throughput.

DAB usually delivers significant performance gains even with lower hit rates than those in the example.

AutoDAB – the autonomous cache manager

The most important aspect of DAB deployment is the decision as to which applications are to be speeded up by DAB and which files or volumes are to benefit from DAB caching.

DAB offers two options for this:

- **Option 1 – Expert mode:** In this case the user personally selects the database to be cached, based on a detailed knowledge of applications and their data access behavior.
- **Option 2 – The self-optimizing cache system:** In this mode, AutoDAB, an intelligent DAB component, independently selects the files to be cached, based on its observations of data flow and cache hit behavior.

In the simplest – self-optimizing – case, the entire available cache memory is configured as one cache area for all performance-relevant disks by system support and selection of the files is left to AutoDAB, which ensures dynamically that frequently used files with good cache hit rates are buffered and files with an unfavorable cache behavior do not use the cache.

AutoDAB – the cache expert

AutoDAB provides a host of functions that internally control and optimize file caching.

The most important of these recognizes sequential access patterns by continuous logging of a number of the most recently executed IOs (Predictive File Table) and uses this as a basis to set the prefetch factor that matches the access profile for the selected files. In order to achieve the biggest possible prefetch value for sequentially processed files, multiple cache segments are filled with data by means of a physical IO (Multi-Segment Prefetch) and simultaneously a predictive asynchronous prefetch is always activated.

The buffered files are cyclically monitored to check their cache usage, i.e. their classification into a specific access profile, and therefore the setting of the prefetch factor, takes place dynamically. Files with a poor cache usage level can forfeit their caching privilege in favor of other files with higher hit rates and their associated applications.

Write IOs to the cache (in write caching) are generally executed very quickly, i.e. the access time corresponds to the time of a read hit. The asynchronous write-back of the cache data combines multiple cache segments into a physical IO and so minimizes disk utilization.

A simple recipe for using AutoDAB

- Select the performance-critical applications
- Select the volumes to be cached containing the data for the applications
- Prepare the cache medium
  - GS : Set up GS partitions for DAB
  - MM: Estimate the available memory
- Set up cache areas and specify the caching method with *BY-CACHE-MEDIUM
- Monitor by means of: /SHOW-DAB-CACHING or SM2 reports (FILE / DAB / PFA / Symmetrix report)

Good reasons for using AutoDAB

- Typically, significant increase in IO throughput
- Short response times for OLTP applications
- Dramatic reduction in long batch running times
- Fast application restart, e.g. databases after unscheduled outages
- Little expert knowledge required

DAB in the HIPLEX

To support caching of shared disks (shared pubsets or SPDs) in a server cluster, DAB provides functions for both read caching and write caching. This always relates to caching of files that are used in the local server environment (this is the default case in file processing).

Read caching of shared disks is supported by AutoDAB in the ADM-PFA concept (DAB commands). For this, cache areas can be set up on a local server basis on the nodes participating in the cluster. DAB guarantees data consistency by invalidating the cache data for the files cached at close time. This ensures that the current data status is accessed following a data update initiated in the interim from another server.

Write caching for shared volumes is supported by DAB only for pubsets in the (User) PFA concept. Here too, local server cache areas are set up on the servers in the cluster.

DAB supports main memory cache areas on the individual servers in the cluster as possible configurations (a CCS cluster is also sufficient for this), as well as cache areas in the local GS (GS only available on one system in the cluster) and in the global GS (GS shared by all cluster participants).

In the case of a globally used GS in the HIPLEX (parallel HIPLEX), functions are available that permit central administration of the GS and its use as a shared medium. In the parallel HIPLEX, the DAB cache areas for shared pubsets reside in a shared GS partition. Such a configuration offers advantages in the event of a server crash, as DAB subsequently reconfigures the affected cache area:

- If a server crashes, automatic failover of the cache areas to the pubset master (or backup master)
- Immediate access to the cache data following failover
- Automatic, asynchronous write-back of the cache data from the switched-over cache area to the pubset incl. dissolution of the (now additional) cache area

DAB caching in the HIPLEX does not just permit use of the GS in the availability cluster, however. DAB caching can also be used to create a high-performance load-balancing cluster. DAB increases the IO performance of the entire cluster.

DAB and file encryption

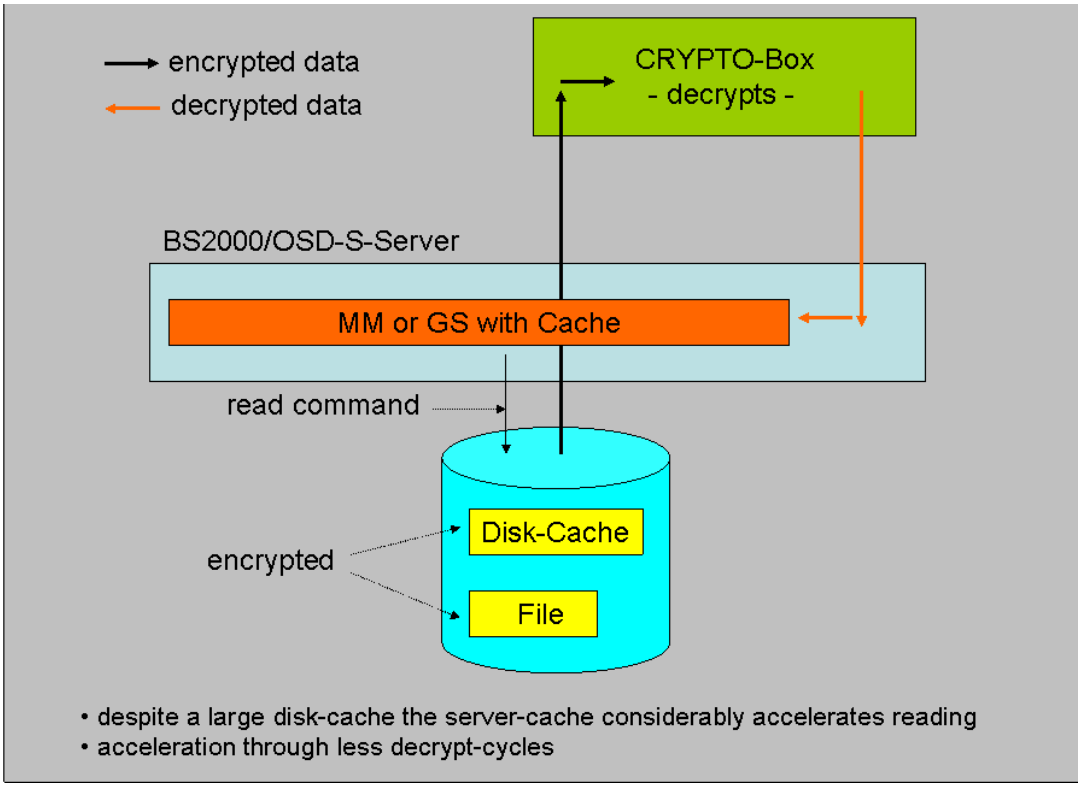
Data from encrypted files must be decrypted before being processed. This is handled by the channel-attached Crypto-Box (S server case) or by a dedicated CPU in the case of SX and SQ servers.

Following the decryption, the data resides in plaintext form in main memory or in a DAB cache. Needless to say, the decryption costs time, because of course, e.g. in the case of the S server, at least two further I/Os to the Crypto-Box are required.

If decrypted data is cached, during read caching with repeated access to the same data there is no need for either the disk I/Os or the two I/Os to the Crypto-Box for decryption purposes.

This gain in performance cannot be achieved by a disk controller cache, since in that case the data is always present in encrypted form.

The following figure illustrates the situation.



Glossary

Term	Definition
AutoDAB	A cache manager that operates like an autonomous system, i.e. it can independently administer caches according to predefined policies. DAB (--> DAB) contains such a component.
Cache	A fast-access memory which contains a partial copy of the data of another, usually slower, memory or storage.
Cache benefits	A cache enables data that is accessed repeatedly to be accessed faster, starting with the second access. The data is fetched from the cache, and not from the downstream, slower memory/storage. A prerequisite for this is that an instance exists which, among other things, monitors the access frequency to data and in the case of frequent accesses stores this "frequently used data" in the cache. --> Cache manager; DAB
Cached file	A file which has been selected for caching by an administrator or by AutoDAB.
Cache hierarchy	A series of caches stored in sequence, which generally behave monotonously descending in terms of access speed and cache costs, but monotonously ascending in terms of cache size (beginning with the fastest cache ...). For example, 1st-level cache --> 2nd-level cache --> controller cache form a cache hierarchy.
Cache hit	When a cache contains the data required by a processor, this is referred to as a cache hit; otherwise it is a cache miss.
Cache manager	A program that administers caches according to administration specifications or policies. -----> DAB
Cache miss	--> Cache hit
DAB (Disk Access Buffer)	A program in BS2000/OSD that administers the caches in main memory and in global storage. --> Cache manager
ILM	Information Life Cycle Management: The idea behind ILM is to manage data during its entire residence time in an IT infrastructure according to predefined usage profiles (policies) in such a way that the total cost for data storage and data provisioning over the entire residence time is minimized (Total Cost of Ownership). Caching is clearly a technical sub-aspect of the implementation of ILM.
L1 cache	Also called 1st-level cache. In a cache hierarchy (--> Cache hierarchy) this is the fastest, most expensive and smallest cache. It is usually accessed by a processor synchronously, i.e. no process or task interrupt is triggered upon a hit (--> Cache hit). L1 and L2 caches are generally implemented directly in hardware, i.e. they are transparent to a cache manager such as DAB.
L2 cache	Also called 2nd-level cache. A cache that is next in the cache hierarchy to the 1st-level cache. It is usually slower and much larger. ---> L1 cache
LRU principle	As a cache is only finitely large, "old" data already present in the cache may have to make way for new data, i.e. this old data is displaced to the next lower level of the cache hierarchy. The LRU principle is based on a replacement strategy whereby the data that has not been used by the processor for the longest (least recently used = LRU) is displaced.
Non-volatile cache	--> Volatile cache
Volatile cache	Volatile caches are caches containing data that may be lost in the event of a server hardware crash, e.g. due to power failure. L1 and L2 caches are generally volatile caches. The Global Store which can optionally be attached to BS2000/OSD servers of the S series, in contrast, can be operated as a non-volatile cache with the aid of redundancy.
Write-in cache	In this cache operating mode, data is initially output only to a cache and is not written to the disk subsystem until later as part of the displacement mechanisms (--> LRU principle). This carries the risk that if the cache crashes, the data will not be consistent on the disk subsystem.
Write-through cache	In this cache operating mode, data is output to a cache and simultaneously written to the disk subsystem.

Published by department:

Frank Schützhold
Phone: ++49 89 3222 2627
frank.schuetzhold@ts.fujitsu.com
ts.fujitsu.com