Firebird Hardware Guide

You can often see the following question in Firebird technical support groups: "What is the hardware of choice for the Firebird DBMS?". This topic remains permanently popular because hardware requirements differ by tasks and hardware itself changes with time.

We decided to write this guide in order to provide the necessary knowledge to anyone who wants to choose truly effective hardware for their Firebird database. To do it, you will have to learn some basic details on how Firebird, the operating system and, of course, hardware functions.

Table of Contents

A Bit of Theory

To find out what hardware will suit your Firebird database best, we have to understand how Firebird uses its components: CPU, RAM, HDD/SSD and how these components interact with the operating system (for instance, with the file cache).

Firebird Server Functional Modules

First of all, we will look into the Firebird functional modules with the help of Figure 1:

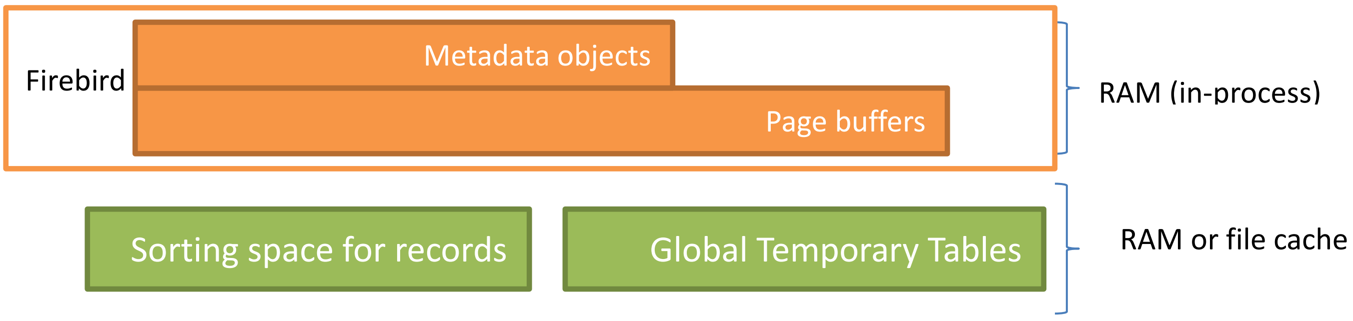


Figure 1. Firebird modules

Firebird includes the following main functional modules:

1) Metadata objects: views, tables, indexes, triggers, stored procedures and other database objects. Metadata objects are located in the address space of the Firebird process (it can be fbserver, fb_inet_server or firebird.exe).
2) The cache of page buffers contains database pages read from the disk and is located in the address space of the server process. The mechanism of caching pages is rather complicated so we will only state that Firebird caches the most often used database pages.
3) Firebird sorts records in memory (in the address space of the server process) until the amount of the memory used for all simultaneous sorting operations reaches the limit set by the TempCacheLimit parameter (firebird.conf). Once this limit is exceeded, a temporary file (with the corresponding operating system flag) is created in the folder with temporary files and it is used for sorting. If there is free RAM in the system, the sorting file will be cached by the operating system and sorting will be carried out in memory.
4) Global Temporary Tables (GTTs) are created as temporary files in the operating system. If the operating system has free memory, operations with GTTs are performed in RAM.

Basic Operations with Hardware

Let us see how Firebird functional modules interact with hardware components during operations performed in the course of working with databases.

Once Firebird is started, the server process occupies the minimum amount of RAM (several megabytes) and does not perform any intensive operations with the CPU or RAM.

When a connection is established to the database, the server starts reading its metadata and create the corresponding objects in memory, which results in the process taking the more resources the more tables, indexes, triggers and other metadata is used. The memory usage increases, but the CPU is practically not used at this stage.

When the client starts executing SQL queries (including stored procedures), the server performs the corresponding operations using hardware. It is possible to single out the following basic operations involving interaction with hardware:

- reading database pages from the hard drive,
- writing database pages to the hard drive,
- reading database pages from the cache,
- writing database pages to the cache,
- reading data from and writing data to global temporary tables,
- processing SQL queries (for instance, JOINs),
- sorting records in resultsets.

Each of these operations requires a certain amount of system resources. The table below shows the resource consumption in intense units (1 means least intense, 10 means most intense):

	Reading a page from the disk	**Writing a page to the disk**	**Reading a page from the page buffers cache**	**Writing a page to the page buffers cache**	**Reading from a GTT**	**Writing to a GTT**	**Sorting records**	**Processing SQL queries**
CPU	1	1	1	1	1	1	5	10
RAM	5	5	5	5	5	5	5	2
Disk I/O	10	10	1	1	1	1	1	1

As you can see, the most resource-consuming operations are those that involve access to the disk because disks remain the slowest hardware component despite the progress during the recent years related to SSDs.

This leads to one of the ways to optimize the performance that is totally hardware-related – take all read-write operations to RAM. But note that the increase-the-page-cache approach does not work. We will deal with this issue in detail in the RAM section.

Simultaneously Performed Operations

Usually, it is necessary to choose hardware for a server that will serve a lot of clients so it is really important to understand how the parallelism of operations is implemented.

From the point of view of hardware components, we can talk about the parallel use of the CPU, the disk and RAM. Modern CPUs have several cores that can execute sets of instructions in parallel so the DBMS server distributes operations between cores, which means the conclusion that the more cores the CPU has, the more clients will be able to work on this server.

It not that simple from the point of view of disks. When traditional hard disk drives (HDDs) read information, they physically move the head over the magnetic material at some finite velocity. A database may be quite large, i.e. 3 terabytes in size, and if SQL queries from clients access its data located in different areas of the disk in parallel, the head of the disk will jump between different areas on the disk thus seriously slowing down the read and write operations. It will considerably increase the disk queue while the rest of the resources (CPU, RAM) are idle. Of course, the disk cache (the cache of the HDD or of the RAID controller) makes up for this slowdown to some extent, but it is not enough.

Unlike traditional HDDs, solid-state drives (SSDs) are much less prone to performance degradation in case of parallel access to data. The advantage of an SSD is especially obvious when you write data in parallel – our tests show that an SSD is 7 time faster than a SATA disk (link!). However, SSDs have some issues that must be taken into account during their use (see Selecting Disks) in order to avoid slowdowns, early breakdowns and data loss.

Operations with RAM are performed very fast on modern computers, they are practically limited only by the data bus bandwidth so these operations do not act as a bottleneck even if there are a lot of parallel SQL queries.

Data Flows

While executing SQL queries, Firebird reads and writes a lot of data, transfers it between functional modules and corresponding hardware components. To identify possible bottlenecks, we need to understand how the data exchange is carried out. Figure 2 below will help us with that:

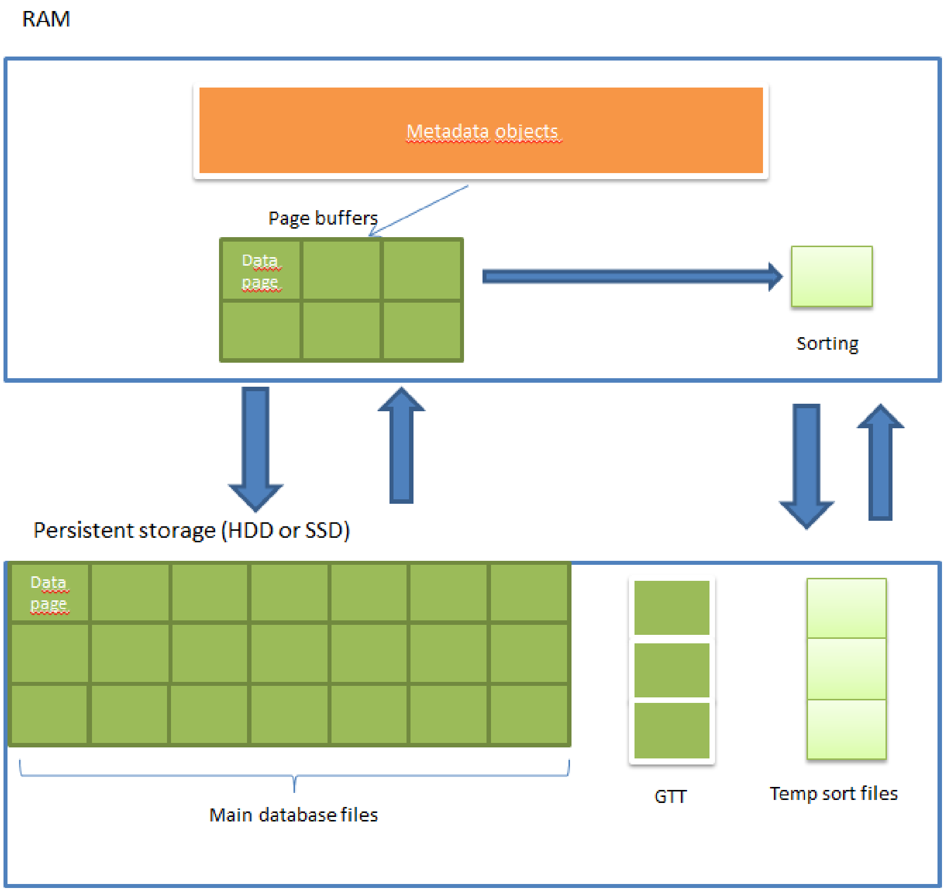


Figure 2. Data flows between RAM and persistent storage

Obviously, transferring data from persistent storage to RAM and back is the most time-consuming operation. It creates two data flows: reading/writing data pages from database files and reading/writing sorting files. Since there can be several sorting files and they can be rather large, they may create quite a heavy load on the disks so it is advisable to direct these input/output flows to different disks.

Backup

Firebird offers you two backup methods: verified backup with the help of the gbak utility and unverified incremental backup with the help of the nbackup utility.

> *We recommend that you combine these backup methods: run nbackup often (for instance, every hour, day and week) and create a verified backup copy every night with the help of gbak.*

Whatever backup method you use, the database file is read (the whole or part of it) and the backup copy (full or incremental) is written. Write operations are performed sequentially during the backup process, which means that regular inexpensive hard drives with the SATA interface (HDD SATA) will be good for backup since they sequentially write pretty fast.

Selecting the Suitable Hardware

Now that we have the idea how Firebird interacts with hardware, we should dwell on the factors that affect the choice of each particular component and its specifications.

Sometimes the actual statistics a particular database strongly affect the choice of hardware components so we will use tools from **HQbird** (the professional distribution package of Firebird from IBSurgeon) to get these statistics. You can download the trial version of HQbird at http://hqbird.com/en/hqbird/.

CPU

While choosing the CPU, you should take into account the following three things:

1) What queries prevail in the application,
2) The number of active connections to the database on average and under peak loads,
3) Firebird version and architecture.

What Queries Prevail in the Application?

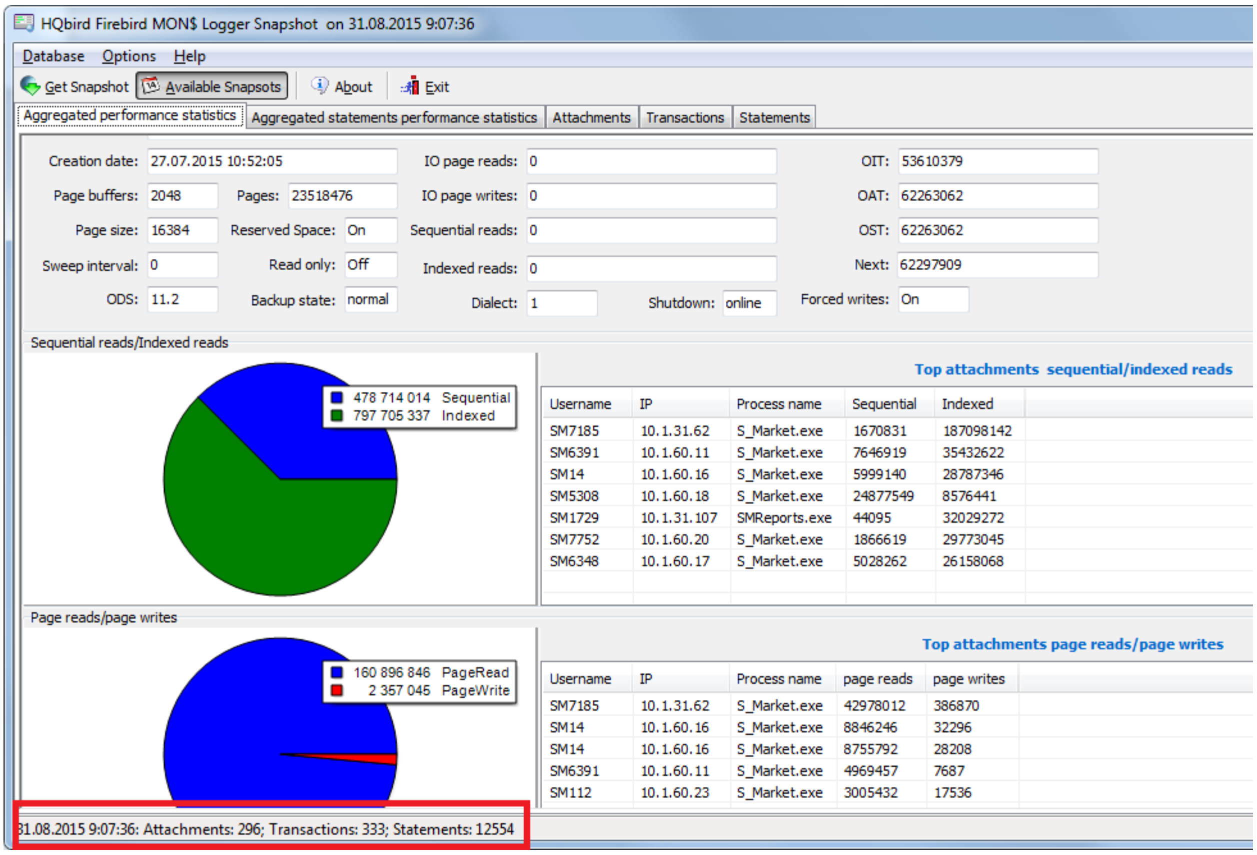
Firebird always executes one query on one core so complex and poorly optimized queries may use one core up to 100% forcing other queries out to less loaded cores and the more cores there are, the lower the chances are that the entire CPU will be used and that users notice any performance degradation in the application.

If the application mainly runs simple short SQL queries, all queries are well optimized and no ad hoc queries are generated (for instance, for reports), the CPU will present no bottleneck for the performance and you can choose a low-end CPU with fewer cores.

If the application contains a report generator or a lot of slow queries returning a large amount of data, you need a CPU with more cores.

The Number of Active Connections to the Database on Average and under Peak Loads

The number of connection (active users) also influences the choice of the CPU. Unfortunately, even application developers have no idea exactly how many connections, queries and transactions are being active at a particular moment. To get more accurate information about this, we recommend that you use the MON$ Logger tool from HQbird and make a few snapshots while it is running where you will see how many connections are actually established.


HQbird Firebird MON$ Logger Snapshot on 31.08.2015 9:07:36
Database Options Help
Get Snapshot
Available Snapsots
About
Exit
Aggregated performance statistics
Aggregated statements performance statistics
Attachments
Transactions
Statements
Sequential reads/Indexed reads
Page reads/page writes

Creation date:	27.07.2015 10:52:05	IO page reads:	0	OIT:	53610379
Page buffers:	2048 Pages: 23518476	IO page writes:	0	OAT:	62263062
Page size:	16384 Reserved Space: On	Sequential reads:	0	OST:	62263062
Sweep interval:	0 Read only: Off	Indexed reads:	0	Next:	62297909
ODS:	11.2 Backup state: normal	Dialect:	1 Shutdown: online	Forced writes:	On

🟦 478 714 014 Sequential
🟩 797 705 337 Indexed

Top attachments sequential/indexed reads

Username	IP	Process name	Sequential	Indexed
SM7185	10.1.31.62	S_Market.exe	1670831	187098142
SM6391	10.1.60.11	S_Market.exe	7646919	35432622
SM14	10.1.60.16	S_Market.exe	5999140	28787346
SM5308	10.1.60.18	S_Market.exe	24877549	8576441
SM1729	10.1.31.107	SMReports.exe	44095	32029272
SM7752	10.1.60.20	S_Market.exe	1866619	29773045
SM6348	10.1.60.17	S_Market.exe	5028262	26158068

🟦 160 896 846 PageRead
🟥 2 357 045 PageWrite

Top attachments page reads/page writes

Username	IP	Process name	page reads	page writes
SM7185	10.1.31.62	S_Market.exe	42978012	386870
SM14	10.1.60.16	S_Market.exe	8846246	32296
SM14	10.1.60.16	S_Market.exe	8755792	28208
SM6391	10.1.60.11	S_Market.exe	4969457	7687
SM112	10.1.60.23	S_Market.exe	3005432	17536

31.08.2015 9:07:36: Attachments: 296; Transactions: 333; Statements: 12554

Figure 3. HQbird MON$Logger: number of connections

For instance, here you can see that the number of connections is 296. Obviously, it is too optimistic to use a quad-core CPU in this case, while a 24-core solution will be quite alright. It is also advisable to count the number of simultaneously running queries since connections may be idle without any running SQL queries.

You can use the rate of 10 to 30 connections per 1 core to roughly estimate the necessary number of cores in your CPU. 10 connections per core for an application with mostly complicated and slow queries, 30 connections per core for an application with mostly simple well-optimized queries.

Firebird Version and Architecture

If you use Firebird version 2.5, note that you should use the Classic or SuperClassic architecture to be able to distribute processing among several cores. In version 2.5, the SuperServer architecture can use only one core for one database so it should not be used in systems that consume a lot of resources.

In Firebird version 3.0, SuperServer, Classic and SuperClassic use the features of multi-core CPUs. Firebird 3.0 SuperServer shows the best performance.

RAM

When you choose RAM, you should pay attention to two things:

1) the memory module must have error-correcting code (ECC RAM)
2) the amount of RAM must be correctly calculated

ECC RAM

ECC RAM considerably reduces the number of errors that occur in the course of working with memory and it is strongly recommended to use it in industrial systems.

Necessary RAM Amount Calculation

To calculate the amount of memory, we will have to look into the peculiarities of various Firebird architectures.

Firebird 2.5 Classic and Firebird 3.0 Classic run a separate process to serve each connection, SuperClassic runs a separate thread for each connection, but practically with the same memory consumption structure – each connection has its own independent page cache.

Firebird SuperServer runs one process with one page cache for all connections.

Thus, the following parameters affect the overall memory consumption:

1) Number of connections
2) Database page size
3) Metadata size (proportional to the number of tables, triggers, stored procedures, etc.; unadjustable; determined by physical usage)
    a. For Classic and SuperClassic – per connection
    b. For SuperServer – per instance of an open database
4) Page cache size (determined by the parameters in the database header or in firebird.conf or in the properties of a particular connection)
    a. For Classic and SuperClassic – per connection
    b. For SuperServer – per instance of an open database
5) Sorting cache size (determined by the parameter in firebird.conf). Note that memory for sorting is allocated not at once but as it becomes necessary.
    a. For Classic – per connection
    b. For SuperServer and SuperClassic – per process (i.e., one sorting cache)
6) For Classic/SuperClassic – lock table size (it is usually small so we will leave it out of our calculation).

The IBSurgeon company performed a few tests and obtained a set of optimal values for the number of pages in the Firebird page cache:

- Classic/SuperClassic – from 256 to 2000 pages
- SuperServer 2.5 – 10000 pages
- SuperServer 3.0 – 100000 pages

Using these tests as a basis, we created optimized Firebird configuration files for servers with c 4-6 GB memory. You can download them here: http://ib-aid.com/en/optimized-firebird-configuration/

Formulas for Calculating the Necessary RAM Amount

Below you can see formulas used to estimate the approximate amount of memory that will be required by Firebird. The actual memory consumption may differ since this estimate does not take into account the amount of memory required for metadata, for the bit masks of indexes, etc., which may increase the memory consumption. However, it is also assumed that the sorting memory will be used to the fullest in all connections, which is usually not the case.
When your database is already in use, you can take a look at the average amount of memory used by the Firebird process (with the help of TaskManager or ProcessExplorer).

Estimate for Classic:

Number of connections * ((Number of pages in cache * Page size) + Sorting cache size)

Example for Classic: suppose we expect 100 active users, the database page size is set to 8 KB and the number of pages in the page cache is set to 256, the sorting cache size is increased from 8 MB (te default value for Classic and SuperClassic) to 64 MB:

100* ((256*8 KB)+64) = 6600 MB

Estimate for SuperClassic:

Number of connections * (Number of pages in cache * Page size) + Sorting cache size

Example for SuperClassic: 100 users, the database page size is 8 KB, the number of pages in the page cache is 256, the sorting cache size is 1024 MB

100*(256*8 KB) + 1024 MB = 2024 MB

Estimate for SuperServer:

(Number of pages in cache * Page size) + Sorting cache size

Example for SuperServer (Firebird 2.5): 1 database, 100 users, the database page size is 8 KB, the number of pages in the page cache is 10000, the sorting cache size is 1024 MB:

(10000*8 KB) + 1024 = 1102 MB

Example for SuperServer (Firebird 3.0): 1 database, 100 users, the database page size is 8 KB, the number of pages in the page cache is 100000, the sorting cache size is 1024 MB:

(100000*8 KB) + 1024 = 1805 MB

"Excessive Memory"

Firebird is often blamed for ineffective memory usage - when the running process of the server consumes a small amount of RAM and the rest of memory remains allegedly unused.

Actually, it is not true. This conclusion basically lies in the misunderstanding of how the Firebird caching mechanism works and in the imperfection of the operating system monitoring tools.

First of all, you have to be absolutely clear that Firebird extensively uses the file cache of the operating system. When a page is loaded into the Firebird page cache, it goes through the operating system file cache. When Firebird unloads a page from its page cache, the operating system keeps holding this chunk of the database in its RAM provided it has enough free memory left.

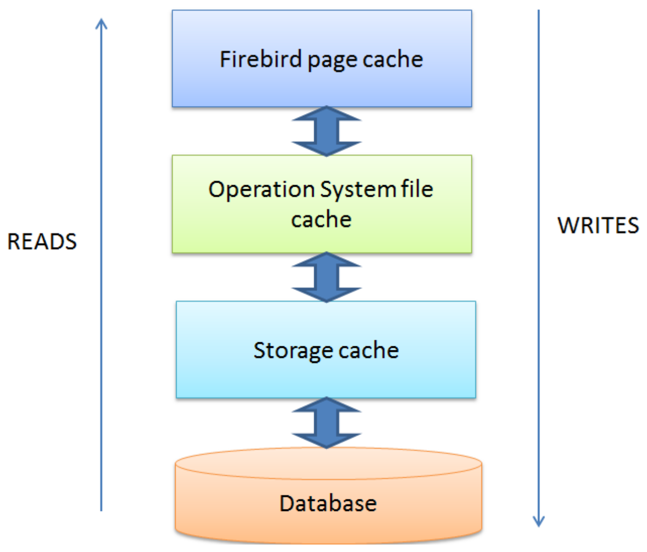
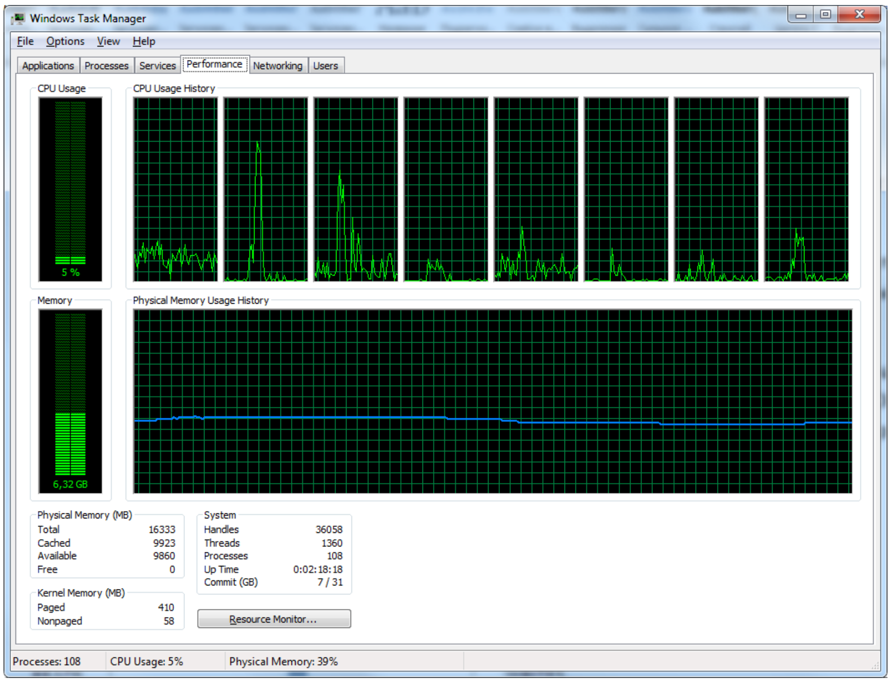


Figure 4. Cache levels: Firebird, OS and storage

However, if you just look at it, the operating system does not show the memory allocated to file cache as being used. For instance, here is the typical situation of memory distribution when the Firebird server is running, as shown by TaskManager:

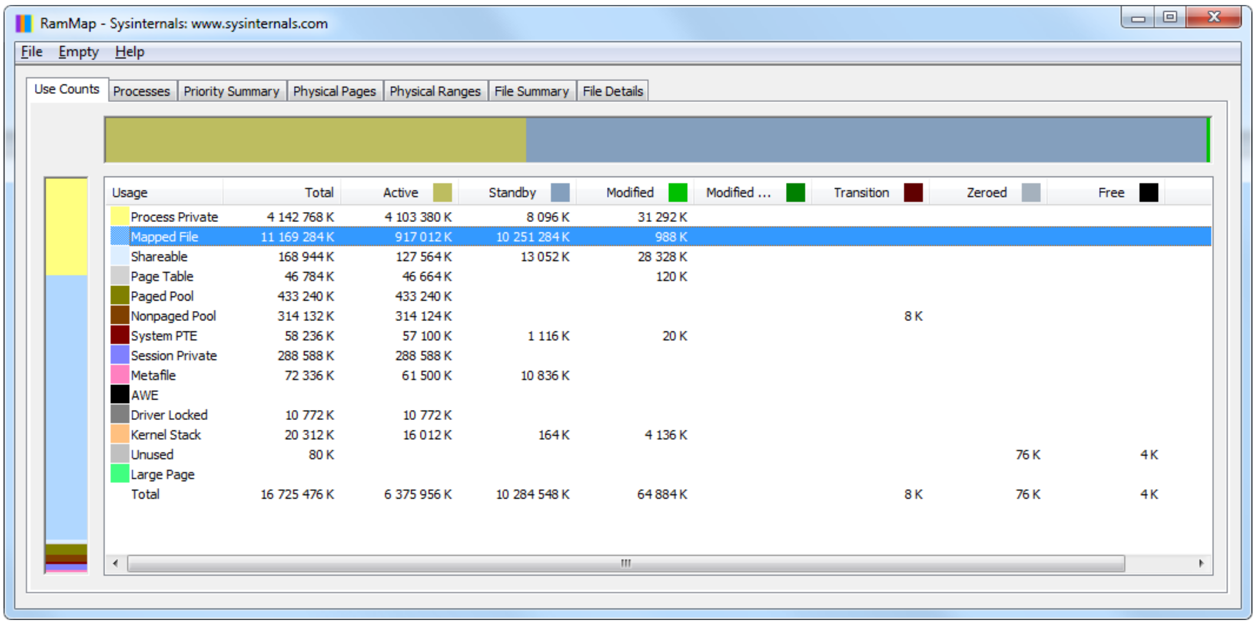


```
Physical Memory (MB)
Total         16333
Cached         9923
Available      9860
Free              0

Kernel Memory (MB)
Paged           410
Nonpaged         58

System
Handles       36058
Threads        1360
Processes       108
Up Time  0:02:18:18
Commit (GB)  7 / 31

Processes: 108    CPU Usage: 5%    Physical Memory: 39%
```

Figure 5. TaskManager does not show file cache usage

It looks as if only 6.3 GB out of 16 GB are used.

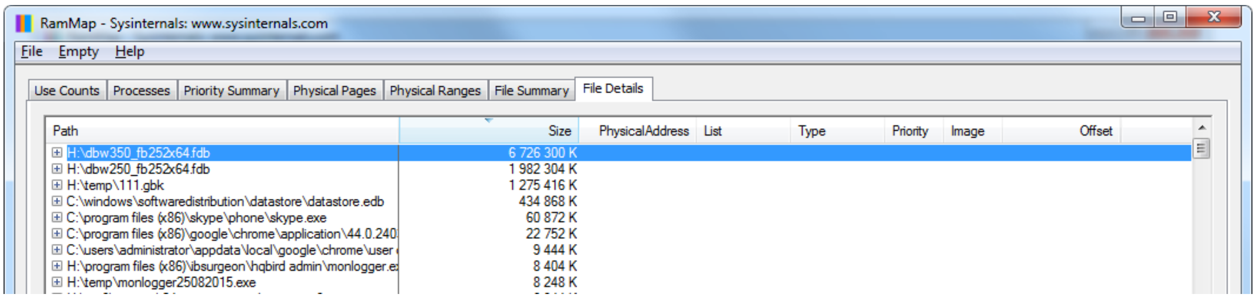
However, if you use the RAMMap tool (from SysInternals by Microsoft), it all looks way more logical:



Usage	Total	Active	Standby	Modified	Modified ...	Transition	Zeroed	Free
Process Private	4 142 768 K	4 103 380 K	8 096 K	31 292 K				
Mapped File	11 169 284 K	917 012 K	10 251 284 K	988 K				
Shareable	168 944 K	127 564 K	13 052 K	28 328 K				
Page Table	46 784 K	46 664 K		120 K				
Paged Pool	433 240 K	433 240 K						
Nonpaged Pool	314 132 K	314 124 K				8 K		
System PTE	58 236 K	57 100 K	1 116 K	20 K				
Session Private	288 588 K	288 588 K						
Metafile	72 336 K	61 500 K	10 836 K					
AWE								
Driver Locked	10 772 K	10 772 K						
Kernel Stack	20 312 K	16 012 K	164 K	4 136 K				
Unused	80 K						76 K	4 K
Large Page								
Total	16 725 476 K	6 375 956 K	10 284 548 K	64 884 K		8 K	76 K	4 K

Figure 6. RAMMap shows details about memory usage: mapped files are cached databases

Database files (dbw350_fb252x64.fdb and dbw250_fb252x64.fdb) are cached by the operating system and occupy the entire memory declared by TaskManager as free:



Path	Size	PhysicalAddress	List	Type	Priority	Image	Offset
⊞ H:\dbw350_fb252x64.fdb	6 726 300 K						
⊞ H:\dbw250_fb252x64.fdb	1 982 304 K						
⊞ H:\temp\111.gbk	1 275 416 K						
⊞ C:\windows\softwaredistribution\datastore\datastore.edb	434 868 K						
⊞ C:\program files (x86)\skype\phone\skype.exe	60 872 K						
⊞ C:\program files (x86)\google\chrome\application\44.0.240	22 752 K						
⊞ C:\users\administrator\appdata\local\google\chrome\user	9 444 K						
⊞ H:\program files (x86)\ibsurgeon\hqbird admin\monlogger.e	8 404 K						
⊞ H:\temp\monlogger25082015.exe	8 248 K						

Figure 7. RAMMap: details about file cache usage

Hence we conclude that the operating system effectively uses the entire available memory to cache the database up to completely loading the database to memory.

Disk Subsystem

The correct configuration of the disk subsystem plays an important role in choosing and configuring hardware for Firebird because any mistakes in this step will result in major failures that are hard to fix.

Separate Disks for Everything

To reduce competition for the disk input/output between operations with the database file and to decrease chances of simultaneous database and backup loss, it is recommended to have three different disks (or raid arrays): one for the database, one for temporary files and one for creating and storing backup copies.

When we say "separate disks", it means that data flows must go through different input/output channels. If you create three logical disks on one physical disk, there will be no increase in performance. However, if you arrange three logical disks on a data storage device equipped with multichannel controllers, the performance will be most likely increased because the device can distribute data flows among controllers.

Sometimes dedicating a separate disk to storing operating system files and the swap file of the operating system is said to increase the performance.

SSD for a Database

SSD is the best choice for working with a database because they ensure great scaling during parallel input/output. It is a must that you use enterprise disks with the increased number of read/write cycles, otherwise it is highly possible that you lose data due to a failure of SSD.

Some time ago, SSDs were prone to increased wear in case there was little free space left on the disk (less than 30%). Simply speaking, each modification on an SSD is written to a new free cell so the lack of free space led to the increased wear of cells that remained free and to the shorter lifespan of the disk.

Manufacturers of modern SSD controllers declare that this problem was solved by preventively moving static data and now the wear of cells is more or less levelled. However, the exact specifications and operation algorithms of SSDs are kept secret by manufacturers so we still recommend that you leave 30% of space on SSDs free as well as reduce their expected lifespan and plan to replace them not less than once in three years.

> *Suppose the size of your database is currently 100 GB, it grows 1 GB per month. In this case, you must not purchase an SSD of minimum size (120 GB), but it is better to choose the next device in the product line - 250 GB. At the same time, buying a 512-gigabyte SSD will be a waste of money since it is advisable to replace the disk in three years.*

The best practice is dedicating an SSD exclusively to working with the database since any input/output operations reduce the lifespan of disks.

Disk for Temporary Files

Since temporary files appear on the disk only when there is not enough amount of RAM, the best way is to avoid this situation altogether, of course. It is possible to evaluate the number and size of temporary files in a production system only by monitoring the folder with temporary files. FBDataGuard from the HQbird distribution package does that kind of monitoring. Once you know how many temporary sorting files are created on the disk and when they are created, you will be able to increase the amount of RAM and change the configuration in firebird.conf.

In any case, Firebird requires that you specify the folder where temporary files will be stored. Usually, the default is left unchanged, i.e. the operating system folder for temporary files is used. If free RAM is enough, this is a good choice.

However, there is another important issue concerning the location of temporary files on the disk – it is creating indexes when you restore a verified backup copy (created with the gbak utility). When an index is created, a temporary file containing all keys from this index is also created. If the database is quite large, the size of the index for some large table can also be quite large. For instance, the index of the largest table comprising 3.2 billion records in a 1-terabyte database is 29 GB, but it took 180 GB of free space to create this index:

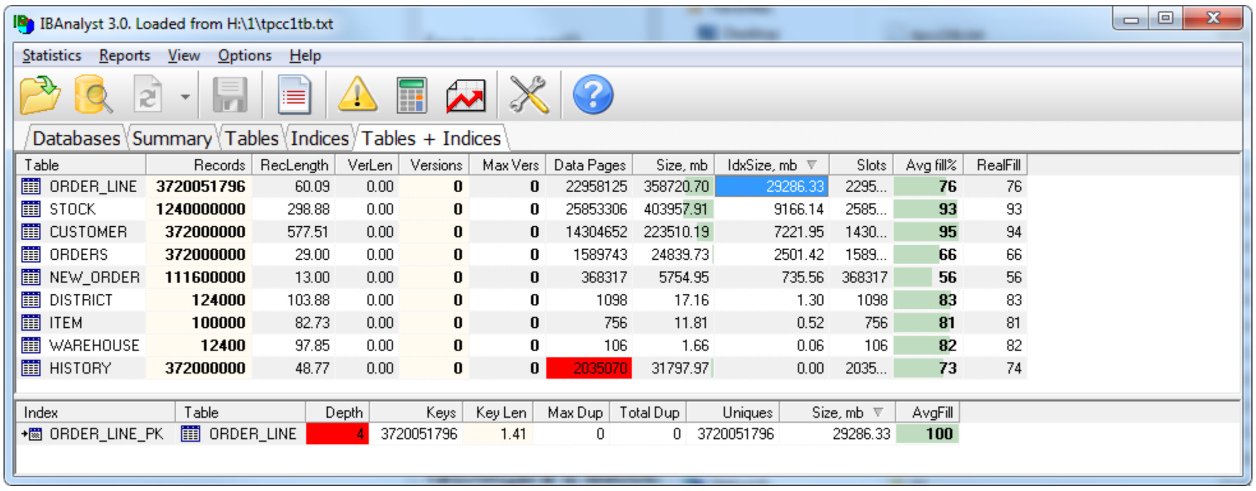


Table	Records	RecLength	VerLen	Versions	Max Vers	Data Pages	Size, mb	IdxSize, mb	Slots	Avg fill%	RealFill
ORDER_LINE	3720051796	60.09	0.00	0	0	22958125	358720.70	29286.33	2295...	76	76
STOCK	1240000000	298.88	0.00	0	0	25853306	403957.91	9166.14	2585...	93	93
CUSTOMER	372000000	577.51	0.00	0	0	14304652	223510.19	7221.95	1430...	95	94
ORDERS	372000000	29.00	0.00	0	0	1589743	24839.73	2501.42	1589...	66	66
NEW_ORDER	111600000	13.00	0.00	0	0	368317	5754.95	735.56	368317	56	56
DISTRICT	124000	103.88	0.00	0	0	1098	17.16	1.30	1098	83	83
ITEM	100000	82.73	0.00	0	0	756	11.81	0.52	756	81	81
WAREHOUSE	12400	97.85	0.00	0	0	106	1.66	0.06	106	82	82
HISTORY	372000000	48.77	0.00	0	0	2035070	31797.97	0.00	2035...	73	74

Index	Table	Depth	Keys	Key Len	Max Dup	Total Dup	Uniques	Size, mb	AvgFill
ORDER_LINE_PK	ORDER_LINE	4	3720051796	1.41	0	0	3720051796	29286.33	100

To prevent the lack of free space on the system disk, it is possible to specify one more disk as additional reserved space in firebird.conf:

TempDirectories =C:\temp; H:\Temp

If there is no space on the first disk, Firebird will continue to use the second disk for temporary files and so on.

HDD for Backups

Regular HDDs with the SATA or nSAS interface will be alright for creating and storing backup copies. They ensure fast sequential write and read operations for backup files and are cheap enough not to save on their size and to keep several backup copies.

Disks for backup copies must always have extra free space left on them: the size of the latest backup copy + 10%. In this case, it is possible to create a fresh backup copy, make sure the backup process is successfully finished (this process may take several hours for a database with a size of several terabytes) and only after that delete the previous backup copy.

If you delete the previous backup copy before the new one is created, it is possible that no new backup copy will be created while the old one will be already deleted and the database will be corrupted, for example, due to a disk failure.

If you use the backup method recommended above (the combination of incremental three-level backup and verified backup once a day storing only one latest copy), use the following formula to calculate the minimum space for backup:

Database_size*3+0.2*Database_size

Let's conside the following sample calculation of space necessary for backup:

Suppose we have a database of 100 GB for which we store three-level incremental backup (week-day-hour - one copy each) and one copy of daily verified backup. In this case, backup copies will take the following space:

- Nbackup_level_0_weekly - 100 GB
- Nbackup_level_1_daily – 5 GB (approximately)
- Nbackup_level_2_hourly – 200 MB (approximately)
- Daily verified backup – 100 GB (approximately)
- Plus you need reserved 110 GB to be able to create the next backup copy.

Total – 316 GB.

! the size of the first-level incremental file or higher depends on the number of pages modified from the moment nbackup was run the last time. The size of these files can be determined only experimentally since the amount of changes in a database depends on applications.

Of course, the estimation of space for backup should take into account the possible abnormal increase in the size of the database and correspondingly increase the amount of free space or otherwise the backup process may be unexpectedly interrupted due to the lack of space.

> *Naturally, smart backup tools (FBDataGuard from HQbird) will notice the lack of space for backup copies and send the corresponding message to the administrator.*

HDD for a Database

An SSD may turn out to be a too expensive solution or the database may be too large and you will have to use less expensive methods.

In this case, you should use an HDD with the SAS interface. If it is not possible, use SATA disks with the nSAS interface or the cheapest option – regular SATA disks.

To increase the speed (and also the reliability – see below) of the hard disks, you should combine them into RAID10. RAID10 is a combination of mirrored (RAID1) and striped (RAID0) blocks. A good and well-configured RAID controller with large cache is a nice alternative to SSDs.

Reliability and RAID

Of course, it is necessary to increase the reliability of the disk subsystem by combining disks in RAID increase in all the variants mentioned above (except for the disk dedicated exclusively to temporary files).

- For SSDs, make sure that you use RAID1 – i.e. two mirrored disks changes are simultaneously written to, which makes chances to lose all data considerably smaller. RAID 10 consisting of SSDs will most likely be redundant since the RAID bus will limit the throughput. For instance, the 6 Gbit/s interface has a throughput of 600 megabytes per second while the modern single SSDs have already reached this speed. Thus, we will get the same limit of 600 MB/s for RAID 10. Except that you can use PCI Express 3.0 to combine SSDs into RAID 10 because the throughput of this bus is already 16 gigabits per second and higher.

- If you use HDDs for backup purposes, it is enough to use RAID1 that will ensure the safety of backup copies and the acceptable read and write speed.
- HDDs used for a database should be combined into RAID10 (at least 4 disks) that provide the optimal combination of cost, reliability and performance. Some users also use RAID5 sacrificing performance for larger space.

RAID Configuration for Firebird

First of all, you should make sure there is a properly charged backup battery unit (BBU) in RAID. If there is no such battery unit, most RAID switch into the safe write mode (the disk cache is completely disabled) that provides lower input/output speed than a regular SATA disk!

> *This fact causes most of frustrated messages to technical support from users who have purchased an expensive server and found out that it works slower than a desktop computer. Unfortunately, some vendors do not include battery units by default that is why it is the first thing that you should check and fix, if necessary.*

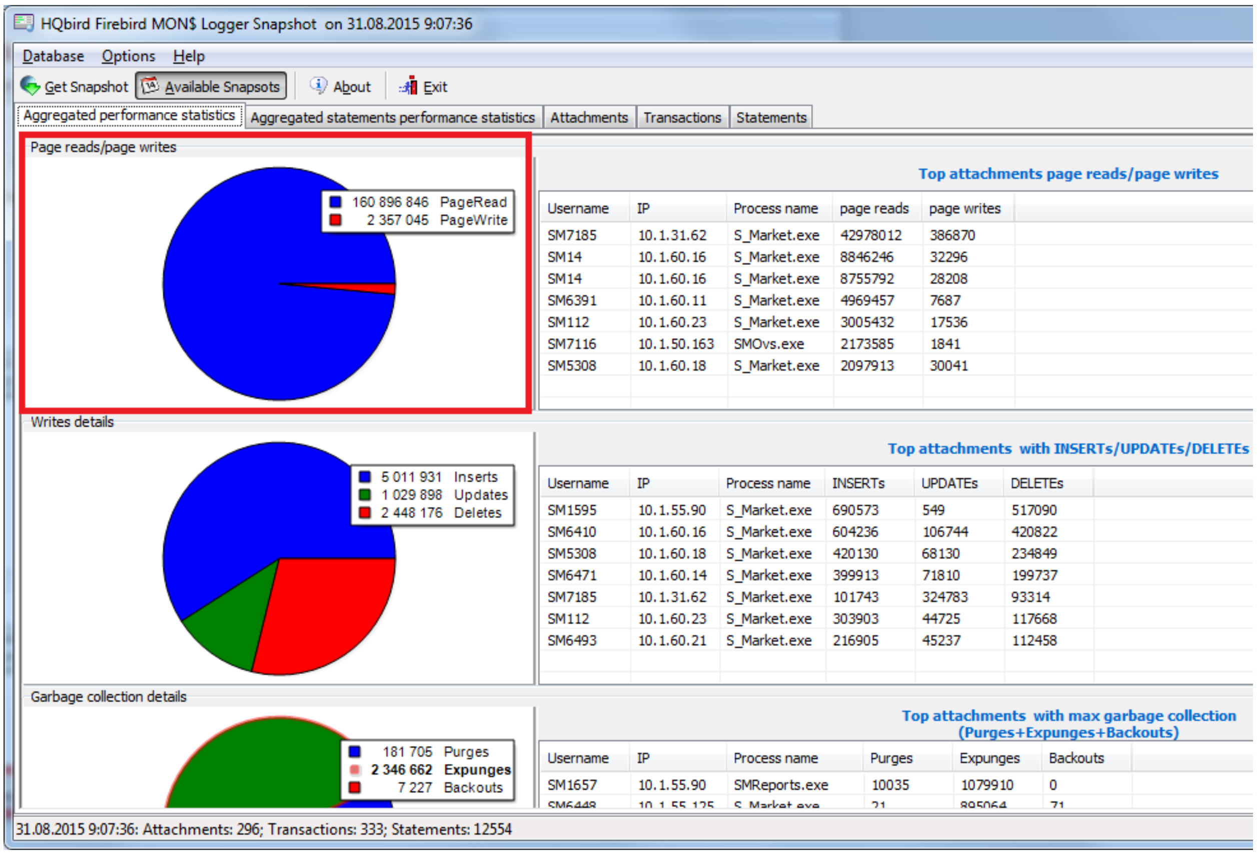
Then you should configure the read and write cache. Often, the cache is disabled by default and if you want to make RAID pretty fast, you need to enable cache.

Besides enabling cache, you should check how it works - it may be write through and write back. The fast way to work with cache is write back - in this case, any changes are written to the cache controller and, in a while, directly to the disk.

You can use tools from manufacturers supplied with RAID to check the battery unit, cache and mode.

Modern RAID controllers can also fine-tune cache – it can be tweaked to facilitate reading or writing. Usually, it is split 50%/50% for reading and writing.

To find out how exactly to configure cache, you can also use the MON$ Logger tool from the advanced HQbird distribution package. It shows the rate of read and write operations to each other (aggregated from the moment of the first connection to the server):


HQbird Firebird MON$ Logger Snapshot on 31.08.2015 9:07:36
Database
Options
Help
Get Snapshot
Available Snapsots
About
Exit
Aggregated performance statistics
Aggregated statements performance statistics
Attachments
Transactions
Statements

Page reads/page writes

🟦 160 896 846 PageRead
🟥 2 357 045 PageWrite

Top attachments page reads/page writes

Username	IP	Process name	page reads	page writes
SM7185	10.1.31.62	S_Market.exe	42978012	386870
SM14	10.1.60.16	S_Market.exe	8846246	32296
SM14	10.1.60.16	S_Market.exe	8755792	28208
SM6391	10.1.60.11	S_Market.exe	4969457	7687
SM112	10.1.60.23	S_Market.exe	3005432	17536
SM7116	10.1.50.163	SMOvs.exe	2173585	1841
SM5308	10.1.60.18	S_Market.exe	2097913	30041

Writes details

🟦 5 011 931 Inserts
🟩 1 029 898 Updates
🟥 2 448 176 Deletes

Top attachments with INSERTs/UPDATEs/DELETEs

Username	IP	Process name	INSERTs	UPDATEs	DELETEs
SM1595	10.1.55.90	S_Market.exe	690573	549	517090
SM6410	10.1.60.16	S_Market.exe	604236	106744	420822
SM5308	10.1.60.18	S_Market.exe	420130	68130	234849
SM6471	10.1.60.14	S_Market.exe	399913	71810	199737
SM7185	10.1.31.62	S_Market.exe	101743	324783	93314
SM112	10.1.60.23	S_Market.exe	303903	44725	117668
SM6493	10.1.60.21	S_Market.exe	216905	45237	112458

Garbage collection details

🟦 181 705 Purges
🟥 **2 346 662 Expunges**
🟥 7 227 Backouts

Top attachments with max garbage collection (Purges+Expunges+Backouts)

Username	IP	Process name	Purges	Expunges	Backouts
SM1657	10.1.55.90	SMReports.exe	10035	1079910	0
SM6448	10.1.55.125	S_Market.exe	21	895064	71

31.08.2015 9:07:36: Attachments: 296; Transactions: 333; Statements: 12554

Figure 8. HQbird MON$Logger: read/write ratio

As you can see, there are much more read operations than write operations in this example so it makes sense to configure the RAID controller for 80% of read operations and 20% of write operations.

SAN and databases

Integrated storages have become popular recently. They include a flexibly customizable array of disks (all types of RAID) with advanced caching features. Usually, SANs have several input/output controllers, which makes it possible to serve several servers simultaneously and work quite fast.

A lot of organizations purchase SANs and use them in their work with Firebird databases. If a SAN is configured correctly, is it possible to achieve good performance. You should take into account the following issues if you use SAN:

1) Several high-performance disk controllers that provide a multi-channel data exchange must be available
2) Backup battery units must be present (BBU) if they are provided by design.
3) Database disks must be combined into RAID10.
4) The cache must be enabled, the write mode must be switched to write back.
5) If several computers are connected to the SAN, each of them must have its own controller.
6) The latest SAN drivers are installed. We have come across cases when later drivers provided gave 30% increase in performance.
7) If there are several logical disks on an SAN (for databases, backup copies, operating system), they have different input/output channels. An attempt to use one channel for all disks altogether will result in lower performance.
8) Similarly, if several servers and databases use a SAN at the same time, the performance may be lower due to the increased bandwidth of input/output controllers.

9) Combined ways are often used - when the operating system and temporary files are stored on local disks while the database and backup files are stored on a SAN.

Often SANs are used as "two servers – one SAN" in order to create a failure-proof cluster. It should be noted that such a cluster can solve problems related only to hardware failures on one of the servers by switching to the second server at once. If the problem is related to the SAN or to the database itself, this solution will not help.

To build an actually failure-proof solution, you should use solutions replicating data between two database instances. You can contact support@ib-aid.com to find out more solutions available for Firebird.

Brief Conclusions and Recommendations

Let us summarize conclusions and recommendations for Firebird concerning hardware.

1) Multi-core CPUs must be used to serve a large number of users
2) The least amount of RAM is calculated on the basis of the number of users and database configuration, the exceeding amount of RAM will be effectively used by the operating system to cache the database file.
3) Use separate disks for databases, temporary files and backup files
4) Rather use SSDs for databases
   a. Reserve at least 30% of free space on SSDs.
   b. It is advisable to dedicate a disk to the database.
   c. Use enterprise SSDs (with a lot of write/read cycles).
5) Make sure you use RAID.
   a. For SSD – RAID 1, for HDD – RAID10, for backup HDD – RAID1.
      i. SAS, SATA, nSAS
   b. Make sure the RAID battery is there and charged
   c. Make sure it is switched to write back.
   d. Some RAID controllers have the cache size already configured, for instance, 75% for reading, 25% for writing, or 50/50 etc. So it is necessary to install MON$Logger - the software that will control RAID parameters, look for the read/write ratio and change the RAID settings.
6) There are pros and cons in using SANs. To get the most of it, you should configure the SAN correctly.
7) To build a failure-proof solution, you need to use solutions with replicates running on different servers.

Contacts

The IBSurgeon/Ibase.ru company have been developing an advanced HQbird distribution package for enterprises, providing a complex technical support for Firebird and developing customized distribution packages as well as solving other complicated issues.

Contact us: support@ib-aid.com, or call +7 495 953 13 34