UNIVERSITY OF GOTHENBURG

Managing memory scalability in web gardens

A study in efficient inter-process memory usage

Stefan Matsson, matsson@ituniv.se
Elias Ung, eung@ituniv.se

Bachelor of Applied information technology thesis

Report No. 2009:039

ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2009

Abstract

With web applications being more and more complex the need for data caching grows larger. When running ASP.NET applications under Internet Information Services in Microsoft Windows it is recommended to run the application in a web farm. This is because it allows multiple requests to be handled simultaneous. Problems emerge when these processes needs to share common data with each other since two processes cannot read each other's memory in Microsoft Windows. Since each process has to allocate memory for the data caching individually this creates a negative impact on scalability. This study concludes that a shared memory solution using memory mapped files is the most efficient solution regarding scalability and overall performance.

Contents

1. Introduction

When running web applications in Windows using Microsoft Internet Information Service [1], one does benefit in performance by running a web application in multiple processes (known as “w3wp processes” [6]), known in server terms as a “web garden”. [2] When doing this the multiple processes can run simultaneously and thereby handle more requests at the same time. This, however, causes problems when the web application needs to load, and keep, data in its memory. Since there are multiple, isolated, processes they cannot share this data with each other. If the web application requires some sort of cache, e.g. to save time processing database requests, this cache must be created by each process and stored in each process’ private memory for fast access. This leads to scalability problems when the number of processes increase.

1.1 Solutions to the problem

When investigating solutions to this problem, no optimal solution that works in all environments and project sizes could be found. There are a number of applications that claim to solve Comfact AB’s problem [3, 4, 5], but these are both too expensive and too complex to put in production at a company of Comfact AB’s size without proper investigation. In this study, the underlying techniques of such solutions will be investigated in order to find out what solution suits companies and project of Comfact AB’s type and size.

1.2 Problem explained

When using Microsoft Internet Information Services (IIS) on a machine that has more than one CPU and/or one than more CPU core, one does benefit in performance by letting IIS operate several w3wp processes. There are two main reasons to do this:

1. Each process can be handled by its own CPU or CPU core which increases the number of requests that the server can handle simultaneously.
2. In 32bit versions of Windows, only 2 GB of memory can be addressed by one single process. This means that if the server has more than 2 GB of memory IIS must run a web garden.

The problem, however, comes when multiple IIS processes must share common data.

When running in a single process, all data that is needed is loaded into that process’ memory and will be kept there until further notice. Since the same process handles the user’s next request, everything will go as planned (see figure 1). However when running a web garden there is no guarantee which process will handle the user’s next request (see figure 2). The problem here emerges since two processes in Windows cannot read and write each other’s memory. When process A generates data and stores in its private memory process B cannot read it and vice versa. This means that when the user switches providing process all data has to be recreated. The time it takes to recreate this data leads to a lot of redundant processing. Each process must therefore keep the cached data for as long as necessary since the speed would decrease too much if the data was to be recreated on every request. The problem that follows is that the memory will be duplicated as many times as there are processes.

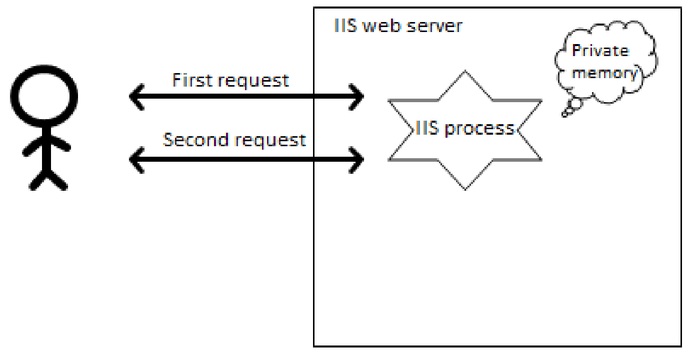

Figure 1 - Single IIS process serving one user

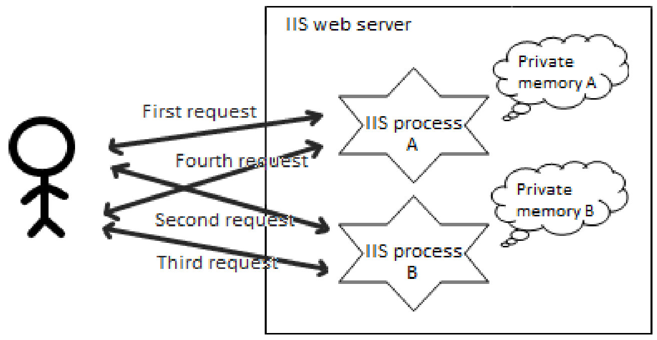

Figure 2 - Multiple IIS processes serving one user

This study will focus on the following research questions:

A. Which are the most efficient ways to share data among processes in our environment?
B. Which solution of the ones found in question A is the best solution for Comfact AB and similar companies considering cost of ownership and usability?

1.3 Motivation

One of Comfact AB's customers was running a web application serving thousands of users simultaneously on a single core server. When all of these users where fetching big data files from a data source the overall performance of the application decreased since it had to recreate the in-memory representation of the data on every page request. To solve this, an in-memory cache was implemented. The problems started again when the customer bought a new server with two dual core CPUs and increased the number of w3wp processes. After the move to the new server, users started reporting problems with slow loading times. When the load of the server reached its peak, with almost 1500 users simultaneously, the server ran out of memory and crashed. The customer was at this time in a very crucial phase of a project and the load of the server was this high for almost one week. The server ran out of memory and crashed every day around 1100 hours (workdays were 0800-1600 hours). When this was further investigated by Comfact AB they discovered that each of the w3wp processes had its own copy of the memory cache, resulting in four times the memory usage that should be needed. The immediate solution was to reduce the number of w3wp processes to a number where the duplicate cache did not use all of the server’s memory. This, of course, decreased performance for the users.

The reader of this article would benefit from our findings if he or she is running a web application, which handles a lot of heavy data, using a web farm. The article can also act as support for companies and organizations that are planning to invest in a multi CPU server and need information on how to best adjust their web applications for the new environment.

2. Literature review

This section contains related research and how the writers of this article’s position themselves towards the related research found.

Mujtaba Khambatti [31] writes about named pipes [8] and sockets under different operating systems. Sockets [9] are a good way of distributing data between applications (and computers). Our problem can be solved using an external server which sends and receives data over sockets. Khambatti’s works lead us to the “Socket server”-prototype. Bagchi and Nygaard [30] also inspired us to on how to use sockets.

Burns and Rees [29] are talking about how to use distributed file systems for sharing data between servers. Our thesis is about sharing data between processes in the same computer but their paper gave us valuable information on what to think about when dealing with the file system. This information lead us to the “File system storage”-prototype.

From Department of Computer Science, City University of London [32], comes a paper on inter-process communication [7] (IPC). The paper talks about that IPC is relatively slow and time consuming and instead it recommends the usage of shared memory, which we used in our “Shared memory”-prototype.

Ingo Rammer has written a book on .NET Remoting [33]. The book describes in-depth what .NET Remoting is, why to use it and how to use it.

Cubrera *et al* [34] has written a paper about web services. Our idea was to use web services in the same way as the socket server. When researching this further we found that web services require the data to be serialized using XML serialization [13] and XML serialization is slow. Another problem with this approach would be that if the web service host is having performance problems, the whole web site would have these problems. Because of this, this technique was not investigated further.

3. Research approach

When choosing a research method for this study, the non-measurable factors provided by research question B encouraged a qualitative approach. In order to answer all research questions, research has to be conducted in different cycles where each cycle provides results to initiate the next. The research method used was Design Science [10] since it is outcome based and suits well when moving from phase to phase iteratively. When a cycle is finished and evaluated, the results will be used in order to initiate the next cycle and when all four cycles are finished, a final result of the study can be evaluated.

3.1 Tools and Techniques

In the first research cycle, the objective was to find what alternatives were available for comparison in order to answer research question A. The goal was to find what techniques are used by solutions today in order to solve redundant memory allocation among processes. In the Related Research section, a number of studies on techniques and solutions were discovered. In this phase of research these techniques and other, less academically documented ones, were identified as they were actually used in the industry today.

This cycle was evaluated by the number of viable solutions found that was later used for comparison against each other. In order to continue to the next cycle, at least two viable solutions were required.

3.2 Prototypes and Implementations

In the second cycle, the objective was to acquire usable prototypes for the techniques gathered in the previous cycle. Those prototypes had to be able to run in the specified test environment for benchmarking against each other. Some techniques found in cycle one were represented only as concepts or insufficiently implemented solutions. In those cases, the first step was to find other implementations using the same technique but with a solution more suitable for our testing purposes. If no such implementations were found, a prototype needed to be developed by us if the technique and its provided research offered enough insight to implement such a prototype. Other techniques were presented only as a part of a larger, multifunctional, enterprise system. In those cases, if the whole system itself was not applicable or available to us, the underlying technique was identified in order to implement our own testing prototype. The specification of the test requirements for the testable prototypes was that they all had the ability to interact with applications written using the .NET framework [11], either natively or using a driver. Our test prototypes all shared support for the following interface:

- Method Initialize with in-parameter “parameters” that initializes the prototype using the parameters provided.

- Method StoreObject with in-parameter 'object' returning the key that the object was stored with.
- Method GetObject with in parameter 'key', returning the object stored with that key.

This interface acted as a wrapper for all the techniques that were investigated in this study and needed to be supported by all the different prototypes. The data being stored needed to be serialized so that it could be passed from one application to another [13]. The prototypes took advantage of binary serialization [12] used in the .NET framework to serialize objects into binary format. The reason for using binary serialization, instead of XML serialization which the .NET framework also supports, in this case was that it was faster [13].

The test environment existed on one of Comfact AB's servers running a 64bit version of Windows 2008 Server and the .NET 3.5 framework.

This cycle was evaluated by the number of prototypes fulfilling the requirements of the test environment presented. In order to move on to the next cycle, at least two prototypes had to be ready for benchmarking.

3.3 Performance benchmarking

The benchmarking cycle focused on fulfilling research question A. The cycle consisted of measuring and comparing the speed and memory scalability of the different testing prototypes. A testing tool was developed in order to test each prototype according to the interface specified in cycle 2. The testing tool was developed by ourselves and focused on storing and fetching data via the selected prototype. The tool ran in many processes and each instance stored the exact same data. The shared data was then read back to all instances of the test tool and the speed of the storing and retrieving was measured as well as how much the amount of system memory increased with every extra process.

When evaluating the prototypes after the benchmark, which was be done by using the Performance Monitor in Windows 7 [16], the main attribute to evaluate was the size of system memory allocated for all test tool processes and the current prototype. As long as the memory scalability was no smaller than the existing scenario, where the total memory size was equal to the number of processes times the size of the data that each process needed to operate, the prototype and underlying technique could be discarded as it does not fulfil research question A. For each prototype that got less memory scalability than the existing scenario, the growth in size of each additional process was weighted in together with the speed performance in order to present the benchmark score for a prototype. When measuring scalability, the private work copy of each w3wp process was not taken into account. This is because Comfact AB had a requirement that none of the existing codebase should be changed and the current codebase requires the process to keep its own private copy while it uses the data. The prototypes that scaled significantly better, with system memory usage per process instance without any major losses in speed, and had no obvious high risk external factors was considered in the next cycle.

3.4 Business compatibility comparison

Cycle four consisted of a series of interviews that revealed what characteristics, besides performance, that is important to Comfact AB's management. With the result from these interviews a new comparison could be done between the top candidates from the performance benchmarking, determining which solution is to be preferred for implementation in the real system.

In order for this cycle to evaluate, one or more possible candidates for use within Comfact AB had to be presented.

4. Prototypes

This section describes the different prototypes that were investigated in this study.

4.1 File system

The file system is widely used by almost every application for persistent data storage. Every document, video, music file etc. found on everyday computers have been saved from an application onto the hard drive via the file system. Different file systems vary in performance and usability, the file system used by Windows in our test environment is NTFS [17]. The .NET framework offers efficient methods for reading and writing data to the file system, these methods were simply wrapped into a very simple prototype.

4.2 Inter-process communication

There are mainly two types of inter-process communication that are used in the industry today; sockets and named pipes. Neither of these techniques is focused on solving our research questions but they can both be used to solve them.

Sockets

Sockets [9] are an old technique used in a lot of applications that communicates over the internal and external networks. Web browsers, email clients, computer games etc all use sockets for communication. To use this technique a server needs to be present holding all the objects, which are later requested by the web application. The disadvantages of having a single back-end server are many:

- If the server goes down, everything is gone. This can of course be solved by using a replicated server.
- Sending each object over sockets from one application to another can be slow if the network is overloaded (only applies when the web application and the socket server are located on different machines) or when the object is very big.

The huge advantage of using sockets as a solution is that the back-end server does not have to written in any language supported by the .NET framework. It does not even have to be running Windows. This could prove important if system written in other languages and for other platforms should be able to access the cache.

The socket prototype was written as a server/client application where the server consisted of one process sending and receiving data to the web applications different processes (i.e. clients). This was because the different processes of the web application are running the same code and because of this we cannot set it to listen, and communicate over, different ports.

Named pipes

The disadvantage of named pipes is that they are stored in a special section of the file system, meaning that this approach is no faster than the file system prototype. Because of this, no prototype will be built using named pipes.

4.4 Shared memory

Sharing memory between applications allow them to access the memory just as if the data was located in the applications private memory. This makes shared memory very fast. In Windows, shared memory is implemented using a technique called “Memory Mapped Files” [14]. MMFs are files located in memory and backed by the Windows page file [15], meaning that files (data objects in memory) that are still active, but have not been used for a while, are saved to the hard drive. When the data is needed again it is loaded into the common memory area, from the page file, where it once again is accessible as any other data in memory.

4.5 Database Cache

The most common approach when is to cache the shared data in a database and let each process read and write the data via a database connection provided by a database driver. Most database management systems on the market also allow a database table to be cached in the system memory rather than on the file system, this would allow even faster read and write times. With proper indexing of the data, this could be a fast and very stable technique.

The Database Management System used for testing this technique was MySQL[18], MySQL was chosen since it is free and has support for functionality such as in-memory tables [19] and other types of memory caching. MySQL also has database drivers ready to use from .NET via ADO.NET [20] which made the implementation of the prototype very easy. The memory storage engine turned out to be quite restrictive since it does not allow columns of ‘text’ or ‘blob’ types [21], these types are crucial in order to store the binary serialized objects that are used by the test program. However, MySQL also supports regular memory caching of a table [22], this allows data to be stored to the file system when the cache size is exceeded, but since the cache size can be configured to a large amount, this technique allows fast access to the shared table.

4.5 .NET Remoting

.NET Remoting [23] is a Microsoft API used for communicating and sharing data between processes in Windows, locally or via a network. .Net Remoting allows an object to be shared as a reference through reference marshalling. This means that every process that uses the shared object would actually use the same object reference, even as the object passes the bounds of the process's application domain [24].

When sharing objects with .NET Remoting, each object that needs to be shared must extend the class MarshalByRefObject [25]. This object is then allocated on a Remoting server component and Remoting client components will access this object via the server. The communication between client and server goes through IPC channels [26], this means that this technique is quite similar to the IPC prototype. The complexity added by the fact that all objects that needed to be shareable required modifications would prove a large problem when implementing this technique in Comfact AB's environment since they have the desire to modify their running code as little as possible. This, in combination with the fact that .NET Remoting uses the same underlying functionality used by other prototypes and the sheer complexity of the API itself, lead to the decision to abort implementation of this prototype before it was ready for benchmarking. It was unrealistic that Comfact AB would ever be able to implement this solution so further testing and investigations were not necessary.

5. Results

This section covers the benchmarking and the results obtained from the benchmarking. The benchmarking was made with two different types of tests. The first test focused on how long time it took for the different solutions to store and retrieve data. The second test focused on memory usage. The tests were conducted using a 981 kilobyte large XML-file that was de-serialized into a class structure in .NET. This class structure was then shared between multiple processes via the different test prototypes, all read and write times as well as memory usage was measured and documented . All times are presented in milliseconds and all memory sizes are presented in kilobytes if nothing else is specified.

5.1 The speed test

The speed test consisted of a single process writing data to the current prototype's test interface. This process represented the process that creates and stores the data cache that multiple processes will use. When the common data was stored to the prototype, five processes attempted to read this data and use it instead of creating its own. Both the read and write times were measured and documented, the scenario was repeated five times in order to receive more accurate numbers.

5.2 The memory scalability test

The memory scalability test consisted of one single process writing the data and 20 different processes retrieving this data. The data was retrieved by all 20 processes with a small delay during a 10 minute session to simulate the load on the server. The memory scalability was measured in

Windows Performance Monitor [16] and documented in order to compare the different prototypes. As comparison the original scalability situation was used (see figure “Original memory scalability” for an example). “Per object” is defined as “per identical de-serialized XML object”. The test scenario was repeated five times.

Figure - Original memory scalability

Number of processes	Memory usage per object	Scalability factor
1	981 KB (1004544 bytes)	1
2	1962 KB (2009088 bytes)	2
5	4905 KB (5022780 bytes)	5
10	9810 KB (10045560 bytes)	10

5.3 File system

Speed

Max write time	Min write time	Avg. write time	Max read time	Min read time	Avg. read time
1744	496	784	249	187	199
624	375	436	218	171	193
577	390	461	808	684	772
670	374	452	1245	860	980
587	385	460	218	187	196

Max write time: 1744
Max read time: 1245
Total average time to write an object: 519
Total average time to read an object: 468

Additional notes:

- The peak-time in the above write test is the result of the file system being very single processes minded, i.e. when multiple processes work towards the file system, the write-times increase a lot.
- The results in this test will be slower over time due to fragmentation of the file system.

Memory scalability

Serialized object: 535 KB (547 857 bytes)
Additional notes: This prototype does not use any memory since all data is stored using in the file system.

Number of processes	Memory usage per object	Scalability factor
1	0 KB (0 bytes)	1
2	0 KB (0 bytes)	1

5	0 KB (0 bytes)	1
10	0 KB (0 bytes)	1

Advantages

- Low memory usage, the latest reads will be cached in memory by the operating system but most of the data will be stored on the hard drive.
- Smaller serialized size than most other prototypes, i.e. the object takes up less storage.

Disadvantages

- Slower the more fragmented the file system gets. Defragmentation is required which wears down the hard drive.
- When hard drive activities increase (e.g. when multiple processes are writing to the disk) the write/read time increase a lot.

5.4 Sockets

Speed

Max write time	Min write time	Avg. write time	Max read time	Min read time	Avg. read time
920	499	627	1139	468	546
1388	483	633	1216	468	681
1123	483	595	1400	468	691
733	499	568	1341	468	565
998	499	572	1372	468	635

Max write time: 1388
Max read time: 1400
Total average time to write an object: 599
Total average time to read an object: 624

Memory scalability

Serialized object size: 1 MB (1048576 bytes)
Additional notes: none.

Number of processes	Memory usage per object	Scalability factor
1	1 MB (1048576 bytes)	1
2	1 MB (1048576 bytes)	1
5	1 MB (1048576 bytes)	1
10	1 MB (1048576 bytes)	1

Advantages

- Can be used across networks.
- Supported on most platforms.

Disadvantages

- High peaks in read/write time.
- Requires an additional server.

5.5 Shared memory

Speed

Max write time	Min write time	Avg. write time	Max read time	Min read time	Avg. read time
218	171	180	218	171	190
171	171	172	218	171	187
187	156	174	220	172	189
187	156	174	218	171	187
187	171	174	202	171	187

Max write time: 187
Max read time: 220
Total average time to write an object: 175
Total average time to read an object: 188

Memory scalability

Serialized size: 535 KB (547865 bytes)
Additional notes: none.

Number of processes	Memory usage per object	Memory scalability
1	535 KB (547865 bytes)	1
2	535 KB (547865 bytes)	1
5	535 KB (547865 bytes)	1
10	535 KB (547865 bytes)	1

Advantages

- Very fast! The studies fastest prototype.
- Consistent read/write times.
- Small object size after serialization.

Disadvantages

- Uses the WinAPI [27], which is a bit complicated to use, to access the shared memory.

4.5 Database cache results

Speed

Max write time	Min write time	Avg. write time	Max read time	Min read time	Avg. read time
686	375	446	234	187	205
624	374	436	249	190	205
577	390	461	235	202	212
670	374	452	250	187	205
671	391	549	249	188	213

Max write time: 686
Max read time: 250
Total average time to write an object: 469
Total average time to read an object: 208

Additional notes: none

Memory scalability

Serialized object size: 1 MB (1048576 bytes)
Additional notes: This solution requires a MySQL server running which requires additional memory. At startup it required 10 MB and when storing 75 objects, of the above size, the MySQL server required approx 100 MB of memory. However, the scalability factor did not increase with multiple processes fetching data.

Number of processes	Memory usage per object	Scalability factor
1	1 MB (1048576 bytes)	1
2	1 MB (1048576 bytes)	1
5	1 MB (1048576 bytes)	1
10	1 MB (1048576 bytes)	1

Advantages

- Much for free. The SQL language gives Comfact AB possibilities to gather objects based on range or e.g. minimum file size.
- One of the studies fastest prototypes on getting data.

Disadvantages

- Requires an additional server.
- Requires additional memory per object after certain amount of objects.

4.6 Benchmark summary

This section gives an overview of all benchmarks performed.

Speed

Prototype	Max write time	Avg. write time	Max read time	Avg. read time
File system	1744	519	1245	468
Sockets	1388	599	1400	624
Shared memory	187	175	220	188
Database cache	686	469	250	208

Memory scalability

Prototype	Memory usage per object	Scalability factor
File system	0 KB (0 bytes)	1
Sockets	1 MB (1048576 bytes)	1
Shared memory	535 KB (547865 bytes)	1
Database cache	1 MB (1048576 bytes)	1

6. Discussion

Since all prototypes used a centralized instance of the shared data that every client process retrieved and used as a working copy, memory scalability turned out to be equally optimal for all implemented prototypes. In terms of speed, performance of all prototypes except the file system and IPC prototypes were sufficient for use in the real system. The two excluded prototypes had too large peaks in read and write times and this would be a problem to put in the production environment.

In the end, the prototype and benchmarking study presented two prototypes that outperformed the others: Shared Memory and Database Cache. These two were both significantly faster and more reliable in performance than the other prototypes and they both would work as an answer to research question A. This left it up to research question B to decide which one was the most optimal to implement into Comfact AB's specific system. When deciding this, the management and development staff at Comfact AB were consulted in order to determine which technique would suit them best. The developers would prefer the Database Cache solution since this would require very little implementation. The database server and communication interface via the database driver was already implemented and tested, it just needed to be installed on the server machine. The management however, was not too keen on relying on third party software. The overhead and unnecessary complexity of a database system together with the fact that all well known and stable database systems required expensive enterprise licenses to run in Comfact AB's environment, motivated the approach to write the server side of this solution themselves. Since they decided to implement the server side themselves, the decision was made to implement this according to the

Shared Memory technique rather than the Database Cache. The strength of the Database Cache was that it required minimal implementation, with the added requirement to implement the server side themselves, Shared Memory is, according to benchmarking, faster and at least equally complicated to implement and integrate into Comfact AB's system.

Conclusion

In this study we wanted to find a solution to Comfact AB's problem (see section "1.2 Problem explained") with memory scalability in their web applications running in web farms. We have shown that all the solutions tested in this study (see section "4. Prototypes") gave a scalability factor of 1 (one) implying that memory usage did not increase with each new process fetching the data. We also proved that a solution based on shared memory using memory mapped files [14] will need the shortest time to write and read objects. A solution of this type might be difficult to implement. Because of this, the second fastest solution (which was the database cache) was also considered for use within Comfact AB. This solution is easier to implement but comes with a lot of unnecessary functionality resulting in both overhead and expensive licenses for third party software where only small parts of the purchased functionality is used.

Comfact AB chose to implement the shared memory solution because of its advantages in speed and simplicity in terms of functionality.

References

[1] - The Official Microsoft IIS Site, http://www.iis.net/, 2009. Last checked: 2009-03-22

[2] - Boosting performance using an IIS web garden | IIS Aid, http://www.iis-aid.com/articles/performance_testing/boosting_performance_using_an_iis_web_garden, 2007-12-13. Last checked: 2009-03-23

[3] ScaleOut StateServer - Product Description, 2008-11-07. Last checked: 2009-03-27, http://www.scaleoutsoftware.com/products/stateServer/index.html

[4] Distributed Caching, ASP.NET Session State, NHibernate Caching, and Caching Application Blocking, http://www.alachisoft.com/ncache/index.html, 2009-03-05. Last checked: 2009-03-27

[5] Oracle Coherence, http://www.oracle.com/products/middleware/coherence/index.html, 2009-03-27. Last checked: 2009-03-27

[6] IIS Worker Process, http://technet.microsoft.com/en-us/library/cc735084.aspx, 2007-12-11. Last checked: 2009-03-30

[7] Interprocess Communications (Windows), http://msdn.microsoft.com/en-us/library/aa365574%28VS.85%29.aspx, 2009-02-12. Last checked: 2009-03-30

[8] Named pipes (Windows), http://msdn.microsoft.com/en-us/library/aa365590%28VS.85%29.aspx, 2009-02-12. Last checked: 2009-03-30

[9] Windows sockets 2, http://msdn.microsoft.com/en-us/library/ms740673%28VS.85%29.aspx, 2009-03-12. Last checked: 2009-03-30

[10] Design Science in Information Systems Research, AR Hevner, ST March, J Park, S Ram - Management Information Systems Quarterly, 2004

[11] .NET Framework Developer Center, http://msdn.microsoft.com/en-us/netframework/default(en-us).aspx, 2009. Last checked: 2009-03-30

[12] Binary Serialization, http://msdn.microsoft.com/en-us/library/72hyey7b(VS.85).aspx, 2009. Last checked: 2009-04-21

[13] Basics of .NET Framework Serialization http://msdn.microsoft.com/en-us/library/ms233836.aspx, 2009, Last checked: 2009-04-21

[14] Managing Memory-Mapped Files in Win32, http://msdn.microsoft.com/en-us/library/ms810613.aspx, 2009. Last checked: 2009-04-28

[15] RAM, Virtual Memory, Pagefile and all that stuff http://support.microsoft.com/kb/555223, 2004-12-12. Last checked: 2009-04-28

[16] Windows 7 Resource Monitor http://www.windows7buzz.com/windows-7-resource-monitor-interface.html, 2008, Last checked: 2009-05-07

[17] New Technology File System http://www.ntfs.com/ , 1998-2009, Last checked: 2009-04-28

[18] MySQL Database Management System http://mysql.com, Last checked: 2009-05-07

[19] MySQL Memory Storage Engine http://dev.mysql.com/doc/refman/5.0/en/memory-storage-engine.html, Last checked: 2009-05-07

[20] ADO.NET http://msdn.microsoft.com/en-us/library/e80y5yhx.aspx, Last checked: 2009-05-07

[21] MySQL BLOB Data Type http://dev.mysql.com/doc/refman/5.0/en/blob.html, Last checked: 2009-05-07

[22] MySQL Query Cache http://dev.mysql.com/tech-resources/articles/mysql-query-cache.html, Last checked: 2009-05-07

[23] Microsoft .NET Remoting – overview http://msdn.microsoft.com/en-us/library/ms973857.aspx, Last checked: 2009-05-07

[24] Application Domains http://msdn.microsoft.com/en-us/library/cxk374d9.aspx, Last checked: 2009-05-07

[25] MarshalByRefObject – MSDN http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject.aspx, 2009, Last checked: 2009-05-12

[26] .NET Remoting IPC Channel http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.ipc.ipcchannel.aspx, 2009, Last checked: 2009-05-12

[2 7] Windows API (Windows), http://msdn.microsoft.com/en-us/library/cc433218(VS.85).aspx , 2009-05-07, Last checked: 2009-05-11

[28] Main memory database systems: an overview, Garcia-Molina, H. Salem, K. Dept. of Computer. Sci., Stanford Univ., CA

[29] *Efficiently Distributing Data in a Web Server Farm,* Randal C. Burns and Robert M. Rees, 2001

[30] *Application controlled IPC synchrony - An event driven multithreaded approach,* Susmit Bagchi, Mads Nygaard, Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU)

[31] *Named Pipes, Sockets and other IPC,* Mujtaba Khambatti, Arizona State University

[32] *Experiences with Distributed Shared Memory,* Systems Architecture Research Centre, Department of Computer Science, City University, London

[33] *Advanced .NET Remoting in VB .NET,* Ingo Rammer, 2002, ISBN: 1-59059-062-7

[34] *Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI*
Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana, IBM T.J. Watson Research Center, IEEE distributed systems online, March–April 2002