Homogenization: A Mechanism for Distributed Processing across a Local Area Network

Mahmud Shahriar Hossain
Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Sylhet-3114, Bangladesh. E-mail: shahriar-cse@sust.edu
M. Muztaba Fuad
Department of Computer Science, Montana State University, Bozeman, MT 59717, USA. E-mail: fuad@cs.montana.edu
Debzani Deb
Department of Computer Science, Montana State University, Bozeman, MT 59717, USA. E-mail: debzani@cs.montana.edu
Kazi Muhammad Najmul Hasan Khan
Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Sylhet-3114, Bangladesh. E-mail: najmul_bd@yahoo.com
and
Dr. Md. Mahbubul Alam Joarder
Institute Of Information Technology (IIT), University of Dhaka, Dhaka-1000, Bangladesh. E-mail: joarder@udhaka.net

ABSTRACT

Distributed processing across a networked environment suffers from unpredictable behavior of speedup due to heterogeneous nature of the hardware and software in the remote machines. It is challenging to get a better performance from a distributed system by distributing task in an intelligent manner such that the heterogeneous nature of the system do not have any effect on the speedup ratio. This paper introduces homogenization, a technique that distributes and balances the workload in such a manner that the user gets the highest speedup possible from a distributed environment. Along with providing better performance, homogenization is totally transparent to the user and user needs no interaction with the system to secure the benefit.

Keywords: Homogenization, Distributed processing, Java, Triangular Dynamic Architecture (TDA), RMI.

1. INTRODUCTION

Triangular Dynamic Architecture (TDA) [10] introduces a mechanism of distributed processing and parallel computation for balancing the workload among the idle machines of a network. The construction of TDA is accomplished by introducing an intelligent server that dynamically categorizes hosts and relates those hosts transparently in a local area network. In a distributed system, there might be thin-clients[7], who possess least processing capability with a minimum resource allotment; also there might be high performance hosts with idle CPU time. All the machines will be properly balanced with equal workload when TDA is applied. When the server finds a job from a client, it divides the job into granules and distributes it to the service providers. After processing, the service providers directly return the outcomes to the requesting client.

An intelligent server must divide the requested jobs efficiently so that the distribution mechanism properly balances the load across the system. TDA provides a dynamic nature of distributed and parallel processing that possesses platform independence and a load-balancing tool called homogenization. Homogenization is a process that assures TDA to balance the workload across a networked environment in a dynamic and intelligent way. Every distributed and parallel processing mechanism suffers a massive problem when the networked environment is a heterogeneous one. Moreover a LAN environment basically encompasses a heterogeneous infrastructure because the hosts vary in hardware architecture, memory, resident Operating Systems, background daemons and many other parameters. Homogenization brings all of these heterogeneous parameters to the same virtual platform. Equal allotment of workload would suffer from speedup degradation with the appearance of a low-performance machine. Homogenization assures speedup even when low-performance machines are involved. It should be implemented in a transparent way with minimum interaction from the user.

2. RELATED WORKS

Although there are several distributed systems [1, 3, 11, 16], there is hardly any work on intelligent job distribution and load balancing.

Scott [14] introduces the basics of client/server computing and component technologies and then proposes two frameworks for client/server computing using distributed objects. The component-based architecture defines the basic preliminary components of TDA. TDA is further developed to communicate among three kinds of hosts - server, client and service-provider. Moreover, TDA establishes dynamic relations on runtime and implements homogenization.

Randall et al. [11] have discussed the scalability of a client server relationship. The distribution architecture is developed turn by turn as the number of clients is increased. The paper describes several existing distributed object oriented systems but they did not show any kind of performance measurement benchmarks against their comments.

Launay et al. [13] introduced a framework that constraints parallelism without any extension to the Java [15] language. The project aims at the automatic generation of distributed code from multithreaded Java programs. Although parallelism is its basic concern, it does not emphasize its performance in load balancing rather it stresses its performance in code generation. In contrast, homogenization enhances parallelism by providing balanced distribution of load among the machines across TDA.

JavaParty [9] transparently adds remote objects to Java by declaration in the source code. It introduces involvement of pre-compiler. It creates multiple Java byte-code files for every single distributable class. JavaParty is specially targeted towards and implemented on clusters of workstations. It combines Java-like programming and the concepts of distributed-shared memory in heterogeneous networks. In contrast, homogenization provides a balancing architecture in TDA with the involvement of an intelligent server without any requirement of pre-compilers. Although JavaParty deals with heterogeneous infrastructure, TDA is enriched with dynamic

homogenization that does not require any static entry about heterogeneous machines.

Another work experimentally compares mechanism of load balancing with existing load-balancing strategies that are believed to be efficient for multi-cluster systems. Nieuwpoort et al. [8] conducted this comparison and established a divide-and-conquer model for writing distributed supercomputing applications on hierarchical wide-area systems. In this research work, an algorithm named "cluster-aware random stealing" is used, which is analogous to homogenization in TDA. But the divide-and-conquer strategy may result in high round-trip time. This is why TDA dynamically uses straightforward homogenization process. Homogenization does not provide only the awareness about the machine-configurations but also it enriches TDA server with the load-information of the hosts.

Fuad et al. [5, 6] introduce a system called AdJava that harnesses the computing power of underutilized hosts across a LAN or WAN. It also provides load balancing and migration of distributed objects through the use of intelligent software agents. Although the migration mechanism used in AdJava is highly automated, it suffers from penalty of migration time of the object. TDA provides mechanism to pass objects to the server and thereafter service providers, but there are administrative preferences that allow real distribution of load through analyzing it entirely or a virtual distribution of load that allows distribution information collection from the server. AdJava uses a simple distribution policy to distribute objects to available machines. If the number of objects to be distributed is more than the number of machines in the system AdJava distributes more than one object to those machines that are loaded lightly compared to other machines in the system. On the contrary, TDA distributes a computation according to the homogenized information about the system. Objects are granulized according to that dynamic information. So there is no need to recycle object-transfer to already loaded service providers by a granule of the same request. AdJava harnesses its performance only through scientific applications while TDA is capable of distributing business applications as well.

3. SYSTEM ARCHITECTURE

TDA is a sophisticated form of client-server relationship that in turn is established over three-tier architecture. Now the classical client server relations are no more suitable [4], applications now follow the three-tier architecture. In TDA, the classical client-server relationship is established dynamically and the three-tier architecture is then merged to it. TDA offers triangular relationships, which is dynamically established by the server. The relationship is constructed between the client, the server and the service-provider. TDA uses Remote Method Invocation (RMI) [16] for implementing the triangular relationships.

3.1 TRIANGULAR DYNAMIC ARCHITECTURE

TDA is called so because multiple triangular relationships are established on demand dynamically at run time. For all of the triangles, the server serves as the common point. The server may also decide to make several triangular relationships against a single request. The relationships also can dynamically switch from one to another, that is, if a service-provider becomes busy after receiving the sub-request from the server, it can send the server a connection refusal request and also sends the current status of the sub-job it was performing. If the server grants the refusal request then the service-provider is free, the server will

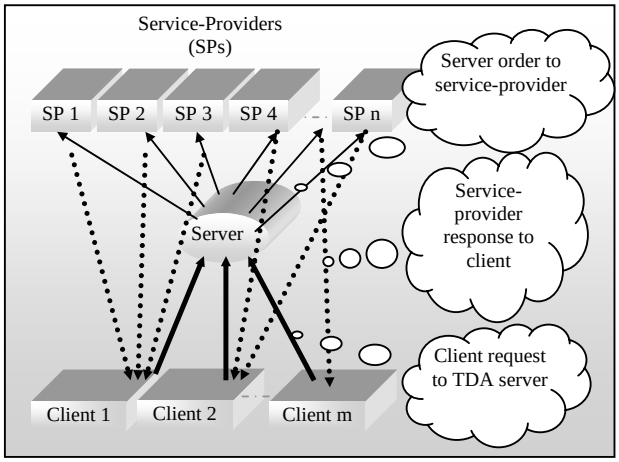

Figure 1: Sample Triangular Dynamic Architecture.

hand over the remaining part of the sub-job to another service-provider that is least busy. It is evident from the Figure 1 that the server is the common point for all the triangles, which means that the server is the one who is responsible for establishing such relations. This is the basic design of TDA. If Client1 sends a request to the server and if the server decides that the request can be divided into three parts, it sends the granulized requests to three service-providers designated as service-provider 1, service-provider 2 and service-provider 3.

The three service-providers process the corresponding sub-jobs in parallel and send the outcomes directly to Client1. In this case three triangular relationships are established, (i) client1, server, service-provider 1, (ii) client1, server, service-provider 2, (iii) client1, server, service-provider 3. For all these dynamically established relationships, the server is the common element, which proves that server is one that is responsible for the decision of distribution.

For the time being, it is assumed that service-provider 1 and service-provider 2 have performance twice than service-provider 3. If the TDA server decides an equal distribution of load to these three service-providers then the distribution would suffer from the problem of parallel processing. The problem is, service-provider 3 would take twice the time taken by service-provider 1 or service-provider 2 for computation. As a result overall computation time becomes a function of the time taken by the slowest machine among the invoked hosts for a particular request. So there should be a mechanism, which would contribute a balanced distribution rather than equal allotment. The distribution should occur in such a fashion that all the invoked service-providers finish their computation at the same time regardless their performance. Homogenization is a process that deals with this problem in TDA.

3.2 TDA SERVER

TDA server is one, which is responsible for the actual distribution of workload. The server maintains some information and based on the stored information, the server can decide about the number of granules to be generated for a particular request. When a request arrives, the server always

depends on the latest data available to its local database; it does not look for more information from the service-providers, since doing so will degrade its performance.

3.3 SERVICE-PROVIDER

Service-providers perform the actual computation in TDA. Background processes are the heart of service-providers. All the processes of a service-provider are hidden from the remote user's sight. A background process always measures the current load of the host even when the service-provider is doing its share of the work. But, it measures its load in such a manner that it does not overwhelm other processes because it is implemented through a low priority thread. Time to time, it communicates with the server mentioning the current load.

3.4 CLIENT

The overall TDA is designed to facilitate the client; to reduce computation time and to perform many jobs that the client alone was unable to conduct efficiently. Furthermore, the client might never perform the job as a thin host. A client program is composed of a user console and a request handler. User console is the basic interface to TDA for the users. If a user casts a request through the console, the request is sent to the request handler. Request handler encrypts the request and sends the request along with the client object reference to the TDA server. The result of processing is received in the user interface portion.

4. HOMOGENIZATION

Figure 2 illustrates the homogenization process for TDA. Java Virtual Machine (JVM) [15] brings all the hosts in TDA to the same platform named homogenization plane. In the homogenization plane all the machines are of same virtual platform but they are of different performance factors. The TDA server performs the next level of homogenization. It brings the service-providers to the homogenization line. This level of homogenization is performed by allotment-variation of workload depending on the performance factors of the service-providers.

In the homogenization line, all the service-providers take same amount of time to complete corresponding sub-requests. Scope length is the length of allotment of workload to a service-provider decided by the server. Scope length variation makes all the service-providers finish their computation at the same time.

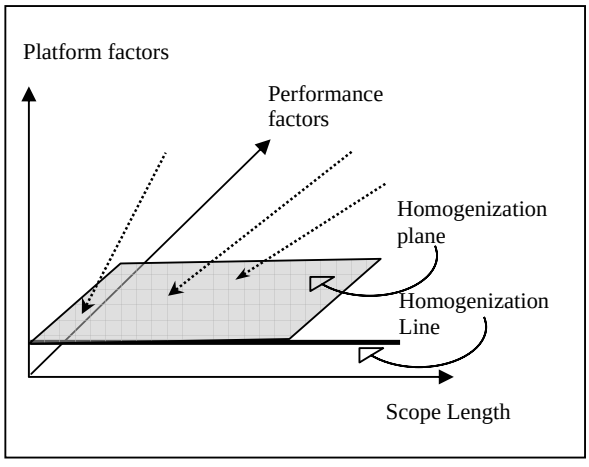

Figure 2: Illustration of homogenization techniques for TDA.

5. HOMOGENIZING TDA

The server maintains several tables in its local database that helps distributing the load. The server actually calculates the scope-length to be offered to a particular service-provider, using the tables of the local database. Most critical knowledge-issues are performance of the service-providers, their response time, list of services provided by a service-provider, etc. A background process in the service-provider informs the server about its current load after every 30 seconds. The server maintains this information and based on the stored information, the server generates a performance number, which is called the homogenized performance. Homogenized performance is the outcome of the second level homogenization of Figure 2. The server depends on the homogenized performance of the service-providers for the balanced distribution of load.

Whenever a service-provider gets an identity during bootstrap, it sends performance parameter to the server. The server also measures the communication distance of the service-provider by pinging it test packets. Time to time, the server upgrades its tables e.g., it sends test packets to get the response time of the service-providers. Test packets are directly thrown back to the server from the service-provider. Moreover, it helps the server to know whether a particular service-provider is dead or active. It helps the server controlling the fault tolerance mechanism. Test packets are smaller in size and they merely congest the traffic. If a service-provider is not busy, but yet it has a large response time, then the server does not invoke it for small jobs. The server always tries to offer it massive and computation intensive jobs so that the time consumed by communication overhead becomes less pronounced.

A service-provider with comparatively lower homogenized performance always gets smaller portions of request than a faster one. A service-provider that is dead with a sub-request keeping it incomplete is again re-requested to another service-provider by the server. This prevents loss of sub-requests, hence the possible loss of client-request. Some service-providers are marked by the administrator as lazy and least busy all the times. Server prefers them as first priority to be involved by sub-requests. The administrator can also set a threshold value for homogenized performance. TDA server ignores service-providers that have homogenized performance less than the threshold value. Therefore, homogenization improves TDA not only as a distributing architecture but also as a sophisticated load-balancing design.

6. PERFORMANCE ANALYSIS

To verify the potentiality of homogenization, a scientific application is implemented in TDA. Performance is measured in two types of environment: heterogeneous environment, and homogenized environment. A homogenized environment is one where TDA has applied homogenization i.e. in reality homogenized environment is a heterogeneous one, but TDA homogenized the overall system.

Matrix multiplication is a common scientific computation that is to be solved for different scientific problems. Considering the simplest algorithm that multiplies two matrices with three loops, the experiment is performed. All the statistics taken are for the same network, same service-providers and the same thin client,

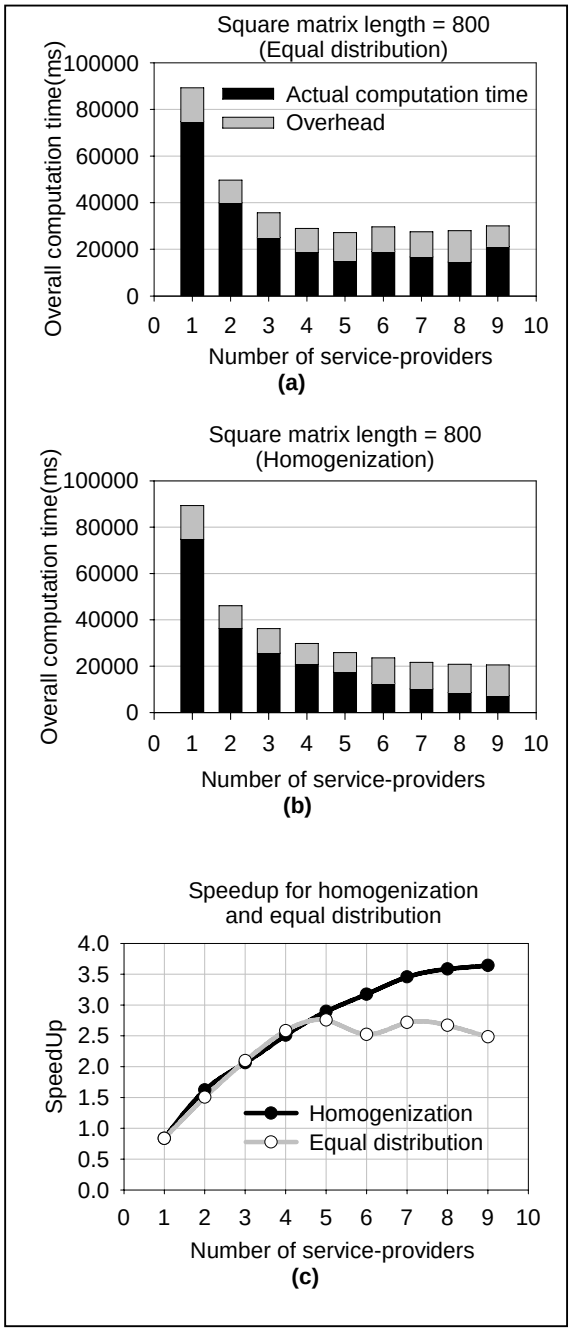

Figure 3: Performance Analysis: (a) Heterogeneous behavior of TDA, (b) Homogenized behavior of TDA. (c) Corresponding speedup of (a) and (b).

as well as the same TDA server. For experimental purpose, the test matrices were all square matrices. Every time two square matrices of same size were requested to the server to distribute. Only the first matrix is granulized into pieces and sent to different service-providers. Each service-provider gets a copy of the second matrix from the thin client. Each service-provider then calculates a portion of the result and sends it directly to the thin client that requested for the job. The thin client combines the result when all the portions are received.

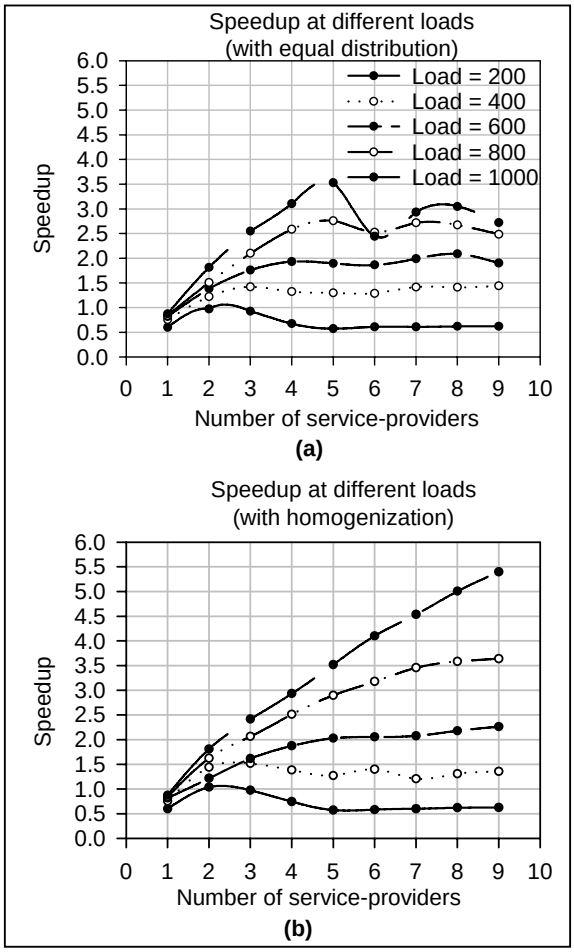

Figure 4: Speedup with different loads for (a) Equal distribution and (b) Homogenization.

The experiments are taken with various combinations of Intel machines. They were varying in CPU speed, memory size, operating system, user processes, background daemons and many other parameters. Pentium II, III, and IV Intel machines with physical memory ranging from 64 to 128 MB are used. All of them are connected by 100 Mbps Ethernet network. All the TDA components were running over the Virtual Machine provided by SUN's JDK version 1.2.2 or higher.

6.1 HETEROGENEOUS BEHAVIOR OF TDA

Figure 3 shows both heterogeneous and homogenized behavior of TDA for square matrix size of 800. The black portion of a bar indicates the actual computation time and the gray portion represents the overhead due to communication distance. From Figure 3(a), it is evident that introduction of successive service-providers reduces the actual computation time. Closer inspection shows that introduction of the sixth and the ninth service-providers do not reduce the actual computation time rather computation time is increased. This type of degradation of performance is found because the sixth and ninth service-providers were comparatively of low CPU speed. Equal allotment of load results in heterogeneous pattern of speedup. The heterogeneous pattern of speedup is shown in Figure 3(c) with a gray line. The speedup pattern shows that speedup is decreased when sixth and ninth service-providers are involved.

Overhead affects speedup because overall computation time is composed of actual computation time and overhead. Overhead is an additive function of communication time and decision making time of the server.

6.2 HOMOGENIZED BEHAVIOR OF TDA

The same analysis is taken with the only exception that now allotment of load is not equal. TDA homogenized the environment. The physical environment is the same as heterogeneous one, but now homogenization is applied. Figure 3(b) shows that application of homogenization assures decrease in actual computation time although the infrastructure is heterogeneous. Corresponding speedup is shown in Figure 3(c) with a black line.

Introduction of newer service-providers causes speed-up improvement regardless their configuration. But the acceleration of speedup is decreased while large amount of service-providers is involved in a distribution. This clarifies that the almost constant overhead becomes pronounced when the actual computation time is reduced. Subsequent involvement of too many service-providers results in slow speedup improvement. In this experiment, homogenization provides a maximum speedup of 3.6 with nine service-providers but non-homogenized distribution provides maximum speedup of 2.8 with 5 service-providers.

6.3 LOAD VS. SPEEDUP

Speedup also depends on the size of the load. Different sizes of matrices are used to understand the behavior. Figure 4(a) shows the speedup lines at different size of matrix multiplication. The figure depicts heterogeneous performance improvement. The matrix sizes are 200, 400, 600, 800 and 1000. For some of the size, speedup is less than unity which illustrates that TDA could not improve the performance because the load was too small. In this case, overheads dominate over the actual computation time. During the size 200, such degradation is found. For all other sizes, speedup is greater than unity. It proves that TDA shows higher performance at higher degree of load.

The corresponding homogenized performance for the same heterogeneous infrastructure is given in Figure 4(b), which shows steady improvement of performance at higher amount of loads. A comparison between Figure 4(a) and Figure 4(b) shows that the maximum speedup reached during non-homogenized situation is around 3.5 where the maximum speedup reached during homogenization is around 5.5 which describes the nobility of homogenization through TDA.

7. CONCLUSION

Homogenization technique based of TDA, is an enriching mechanism of job distribution across a local area network. TDA granulizes computation intensive jobs to concurrent pieces using homogenization and operates them in a dynamic environment to reduce total processing time. Experimental analysis shows that in a heterogeneous environment, homogenization provided a 55% increase in speedup relative to maximum non-homogenized performance. Homogenization does not require any kind of user interaction for its knowledge-centric distribution mechanism. It is established with an automatic manner in TDA as a transparent load-balancing tool.

Homogenization provides better processing time in a distributed computing environment. For implementing homogenization, the present JVM remains unchanged. The current implementation is fully based on the existing JVM and that way TDA fulfills its main goal of providing a distributed computing environment in an existing LAN.

8. REFERENCES

[1] Baratloo A., Karaul M., Kedem Z. and Wyckoff P., "Charlotte: Metacomputing on the Web", *Proceedings of the 9th Conference on Parallel and Distributed Computing Systems*, 1996.

[2] Carnegie Mellon Software Engineering Institute, "Three Tier Software Architectures", http://www.sei.cmu.edu/str/descriptions/threetier.html.

[3] Christiansen B., Cappello P., Ionescu M.F., Neary M. O., Schauser K. and WU D., "Javelin: Internet Based Parallel Computing in Java", *ACM 1997 Workshop on Java for Science and Engineering Computation*.

[4] Collet Christine, "The NODS project, Networked Open Database Services", *Proceedings of Symposium on Objects and Databases (ECOOP)*, http://www-lsr.imag.fr/ Les.Personnes/Christine.Collet, LNCS 1813, Sophia Antipolis and Cannes, France, June 2000.

[5] Fuad M. M. and Oudshoorn M. J., "AdJava - Automatic Distribution of Java Applications", *Australia Computer Science Communication*, Vol. 4, No. 28, Page 65-77, February 2002.

[6] Fuad M. M. and Oudshoorn M. J., "Automatic distribution and load balancing of Java objects in an agent oriented distributed system", *ICCIT, Proceedings of 5th International Conference on Computer and Information Technology*, Page 101-107, Dhaka, Bangladesh, December 2002.

[7] Jennings T., "Application Deployment and Integration", *Research Paper: Jacada ™ Ltd. Jacada ® for Java*, http://www.jacada.com/products/JacadabyButler.pdf, March 2001.

[8] Nieuwpoort R. V., Kielmann T. and Bal Henri E., "Efficient load balancing for wide-area divide-and-conquer applications", *ACM SIGPLAN Notices*, Vol. 36, No. 7, Page 34-43, July 2001.

[9] Philippsen Michael and Zenger Matthias, "JavaParty – Transparent Remote Objects in Java", *Concurrency: Practice and Experience*. Vol. 9, No. 11, Page 1225-1242, November 1997.

[10] Hossain, M.S. and Khan, K.M.N.H., "Triangular Dynamic Architecture", *4th Year project report*, Dept. of CSE, Shahjalal University, Sylhet, Bangladesh, June 2003.

[11] Hyde R. L. and Fleisch B. D., "A Case for Virtual Distributed Objects", *Int'l Journal on Parallel and Distributed Computing*, Vol 1, No. 3, September 1998.

[12] Izzat M., Recht T. and Chan T., “Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications”, *ACM 1999 Java Grande Conference*.

[13] Launay P. and Pazat J. L., “A Framework for Parallel Programming in JAVA” *High-Performance Computing and Networking, International Conference and Exhibition*, Page 628-637, Amsterdam, The Netherlands, April 1998.

[14] Scott M. Lewandowski, “Frameworks for Component-Based Client/Server Computing”, *ACM Computing Surveys (CSUR)*, Vol.30, No.1, Page 3-27, March 1998.

[15] Sun Microsystems, “The Real-Time Specification for Java”, www.java.sun.com, 2003.

[16] Sun Microsystems, “Java Remote Method Invocation Specification”, www.java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html, 2003.

[17] Yarrington P., Collier C. and Pickenheim M., “Parallel Processing with HyperSizer”, *White Paper, Collier Research Corporation*, http://www.collier-research.com/pdf/wp01_parallel_processing_with_hypersizer.pdf, September 2001.