HOMOGENIZATION FOR DIVISIBLE AND NON-DIVISIBLE PROBLEMS ACROSS A LOCAL AREA NETWORK

MAHMUD SHAHRIAR HOSSAIN[1], M. MUZTABA FUAD[2], MD. MAHBUBUL ALAM JOARDER[3]
[1]*Lecturer, Department of Computer Science and Engineering*
Shah Jalal University of Science and Technology, Sylhet-3114, Bangladesh.
E-mail: shahriar-cse@sust.edu
[2]*Department of Computer Science*
Montana State University, Bozeman, MT 59717, USA.
E-mail: fuad@cs.montana.edu
[3]*Assistant Professor, Institute Of Information Technology (IIT)*
University of Dhaka, Dhaka-1000, Bangladesh.
E-mail: joarder@univdhaka.edu
Fax: +880-2-8615583

Abstract:
A networked environment suffers from unpredictable speedup behavior during distributed processing due to heterogeneous nature of the hardware and software in the remote machines. It is challenging to get 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 illustrates homogenization, a technique that balances and distributes a divisible workload in such a manner that the user gets the highest possible speedup from a distributed environment. Besides, homogenization ensures minimum processing time for non-divisible independent jobs. Homogenization is totally transparent to user and system.

Keywords:
Homogenization, Distributed computing, Triangular Dynamic Architecture (TDA), Load balancing.

1. Introduction and Background

Distributed computing in a LAN environment uses heterogeneous infrastructure where hosts vary in processing power, hardware architecture, memory, resident operating system and background daemons etc. Simply allotting equal amount of workload to each machine would cause degradation of speedup for low-performance machines. Only few of the proposed distributed computing operates efficiently in such heterogeneous environment. Moreover as speedup is always dependent on actual computation time and the overhead for communication, a profound relationship between load, computation and data to be transferred is most essential. By establishing this relationship, homogenization [7] enriches communication abstraction. Homogenization is applicable not only to linearly divisible jobs, also to non-divisible problems that are running independently in the remote machines.

System such as AdJava[3, 4] harnesses the computing power of underutilized hosts across a LAN or WAN, provides load balancing but suffers from penalty of migration time of the object and is useful only for scientific applications. Nieuwpoort et al.[5] established a divide-and-conquer model for writing distributed supercomputing applications on hierarchical wide-area systems. But the divide-and-conquer strategy may result in high round-trip time. Although there are several distributed systems[1, 2, 7, 9], there is hardly any work on intelligent job distribution and load balancing.

Triangular Dynamic Architecture (TDA)[6] introduces a mechanism of distributed processing and parallel computation for balancing the workload among the idle machines of a network. An intelligent server divides the job efficiently so that distribution mechanism properly balances the load across the system. TDA provides distributed and parallel processing, a dynamic nature with a load-balancing tool called ***homogenization*** and

possesses platform independence. Homogenization enables TDA to balance workload across a network in dynamic and intelligent way. Homogenization considers the heterogeneous parameters of a LAN and allocates workload to clients based on their capability. Thus, Homogenization avoids degradation in speedup caused by low-performance machines.

Homogenization can be applied to any kind of linearly divisible distributed processing system and non-divisible independent parallel processing environment. It can also be used for solving computation intensive scientific problems. Commercial applications, where faster manipulations are required and thousands of clients are to be served, homogenization would provide better performance.

2. System Architecture

Homogenization is a mechanism that is established over Triangular Dynamic Architecture (TDA) which possesses the components TDA server, service provider and clients. Homogenization facilitates TDA in distribution of workload in a load-balancing manner.

Figure 1 illustrates the homogenization process for TDA. Java Virtual Machine (JVM)[9] brings all the hosts in TDA to the same platform named homogenization plane. This is the first level of homogenization. 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 of variable 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 workload to a service-provider decided by the server. Scope length

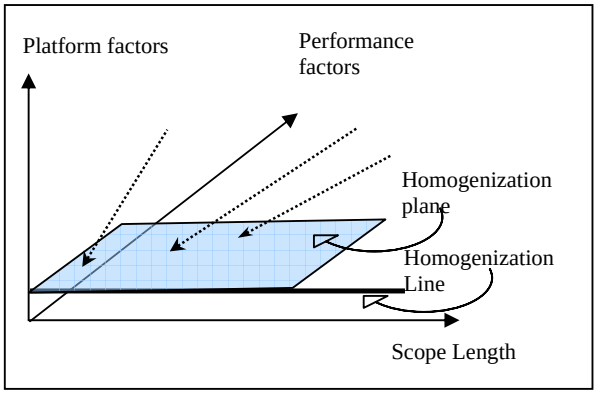

Figure 1: Illustration of homogenization techniques for TDA.

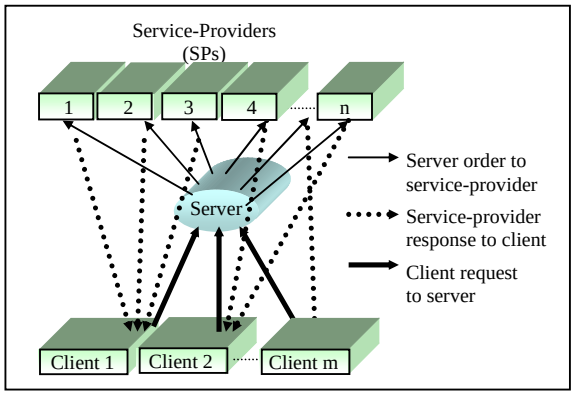

Figure 2: Triangular Dynamic Architecture.

varies based on performance factors of service-providers so that they finish their computation at the same time.

If the requests were divided into equal pieces, slower service providers would take more time than a faster one. Homogenization overcomes this problem by using intelligent load balancing. The distribution of job depends on a dynamic collection of performance report from the service providers. The server uses *maXAgent* strategy for distribution of non-divisible independent jobs.

Homogenization promotes the way of establishing triangular relationships in TDA. The client sends request to the TDA server, the server thereafter granulizes the request into subparts and sends them to service-providers. The server maintains several tables in its local database. It calculates the scope-length for a particular service-provider, using these tables. 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 certain time interval. The server stores these information and based on these generates a performance number called the ***homogenized performance value***. Homogenized performance is the outcome of the second level homogenization of Figure 1.

The first level of homogenization, i.e., bringing all the machines to homogenization plane is done by JVM. TDA server, by generation of homogenized performance value for each machine, does the second level of homogenization, thus bringing all the service-providers to homogenization line. Figure 2 illustrates a sample of TDA that shows multiple triangular relationships established against a request from the client. The relationships are established dynamically based on the homogenized performance of the service-providers.

2.1 Mathematical synopsis for Divisible Problems

Let the time to complete a job in a standalone machine be ***T***; if there exists ***N*** computers those have the same performance along with same homogeneous behavior as the standalone one, then total time to compute the same job with ***N*** computers will be

$$T_N = \frac{T}{N} \quad \cdots\cdots(1)$$

In fact, this is the best theoretical case where the distribution mechanism itself does not carry any cost of distribution. If the cost of distribution is ***O(L)*** where ***L*** is the amount of load that is to be distributed among ***N*** computers, total time to compute the same job would be

$$T_N = \frac{T}{N} + O(L) \quad \cdots\cdots(2)$$

So far, the mathematical synopsis is based on a homogeneous environment. But a real networked environment is composed of heterogeneous elements. TDA server depends on the performance values of the service-providers for a balanced distribution. As a result the concept of "number of computers" becomes somewhat different. The number of computers depends on two parameters. They are: $\boldsymbol{P_S}$, the performance of the standalone machine and $\boldsymbol{P_T}$, the sum of performance values of all the machines invoked by the distribution. That is,

$$P_T = \sum_{i=1}^{N} P_i \quad \cdots\cdots(3)$$

Hence, although physically the number of computers is ***N***, during homogenization virtually the number of computers would become,

$$N_H = \frac{P_T}{P_S} = \frac{\sum_{i=1}^{N} P_i}{P_S} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Thereafter for homogenization in a heterogeneous infrastructure Eq. (2) would get the following form:

$$T_{NH} = \frac{T}{N_H} + O(L) = \frac{T}{\frac{\sum_{i=1}^{N} P_i}{P_S}} + O(L) \quad \ldots\ldots\ldots\ldots (5)$$

Naturally, speedup for a homogenized system can be defined as

$$S_{NH} = \frac{T}{T_{NH}} = \frac{T}{\frac{T}{\frac{\sum_{i=1}^{N} P_i}{P_S}} + O(L)} \quad \ldots\ldots\ldots\ldots\ldots\ldots (6)$$

The distribution overhead ***O(L)*** would become negligible for high degree of computation where actual computation time, $\frac{T}{N_H}$ plays the most dominating role. In this case,

$$T_{NH} = \frac{T}{N_H} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

and

$$S_{NH} = \frac{\sum_{i=1}^{N} P_i}{P_S} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

This case would lead to a linear increment of speedup in accordance with $\boldsymbol{N_H}$, the virtual number of computers.

Overhead ***O(L)*** is not only a function of communication time but also it depends on the time to deliver the information about the network. As the information is stored in the server, the manipulation of the distribution of objects takes a very low amount of time compared to the communication time. As a result overhead is a very linear function of only the load i.e., ***O(L)***.

$$O(L) = M \times L \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (9)$$

M, the slope of the line depends on the infrastructure of the local are network, to which the system is established.

2.2 Synopsis for Non-divisible Problems

The goal of this paper is not to harness the computation power with the best possible solution of a problem, but to derive a distribution mechanism that would result in fast solution in an existing parallel system.

An existing system can be considered as if it is running parallel agents in remote hosts in such a manner that all the hosts are searching for a solution of a problem with some probability. If a solution is found, TDA server would stop all the agents accordingly. Let it be considered that each agent is basically a thread pursuing a search heuristic starting from a random position. More threads on a processor would mean more computation time for a given number of state evaluations per thread. Counteracting this, multiple threads increases the number of random explorations, increasing the likelihood that some thread will lead to a solution. So, there should be some synopsis with the number of agents running in each machine. This type of system that holds independent agents, which are randomly initialized, but concurrently doing the same job in the same space should be rearranged by number of agents in each remote machine. The parameter, *maXAgent* is a system-performance dependent integer that defines the number of agents in a machine where the solution is retrieved in the shortest possible time. It should be mentioned that each service provider should hold *k* agents where *k* is an integer that is equal to *maXAgent* for corresponding machine-performance. From detailed experiments and analysis it is found that *maXAgent* varies system to system depending on the performance of the hosts. The variation follows Equation (10).

$$maXAgent = \frac{7000 - e^{p}}{1000} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (10)$$

where p = performance of the machine in hundred thousand keys per second. A machine performance is taken by key encryption capability of the machine per second. The experiments are described in Section 3.2.

3. Performance Analysis

To verify the potential of homogenization, some problems are implemented in TDA. A linearly divisible problem and a non-divisible instance are taken for the performance analysis.

3.1 Linearly Divisible Problem

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 it is a heterogeneous one but TDA has homogenized the overall system.

Matrix multiplication is a common scientific computation that is linearly divisible in a distributed computing system. 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, 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.

The experiments are taken with various combinations of Intel machines 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 were used. All of them were 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.

3.1.1 Heterogeneous behavior of TDA

Figure 3 shows both heterogeneous and homogenized behavior of TDA for square matrix size of 800. The dark portion of a bar indicates the actual computation time and the gray part 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 does 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.

3.1.2 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 same as the 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.

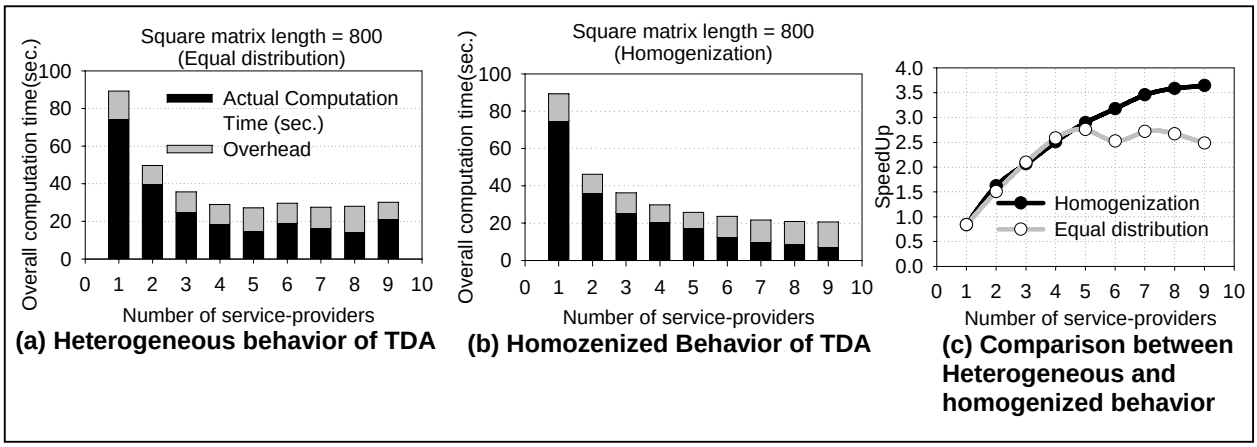

Figure 3: Performance Analysis for homogeneous and homogenized behavior.

Figure 4 compares the same homogenized behavior of Figure 3 with formulae of section 2.2. The summation of performances of the machines is taken from a single snapshot. But runtime performance varies during operation. Moreover the overhead may vary time to time due to congestion in the network. As a result, a deviation from the formula is found for the speedup. The overhead function is a linear one. ***M***, of Eq. (9) is 20 for our network. It can vary network to network depending on the speed of the network.

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.

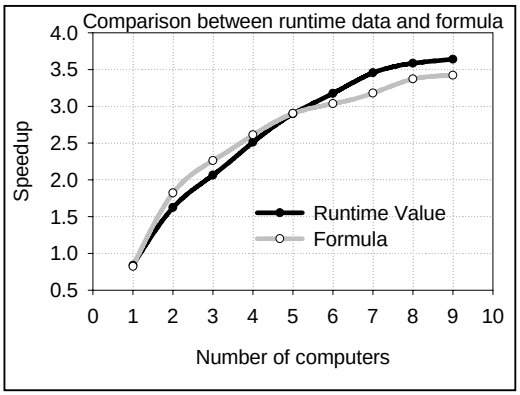

Figure 4: Performance analysis for homogenized behavior of TDA and performance by formula.

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, Figure 3(c) shows that 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.

3.1.3 Load and linearity

Speedup also depends on the size of the load. Different sizes of matrices are used to understand the behavior. Figure 5(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 sizes, 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. Such degradation is found for the size 200. For all other sizes, speedup is greater than unity. It proves that TDA shows higher performance at higher degree of load.

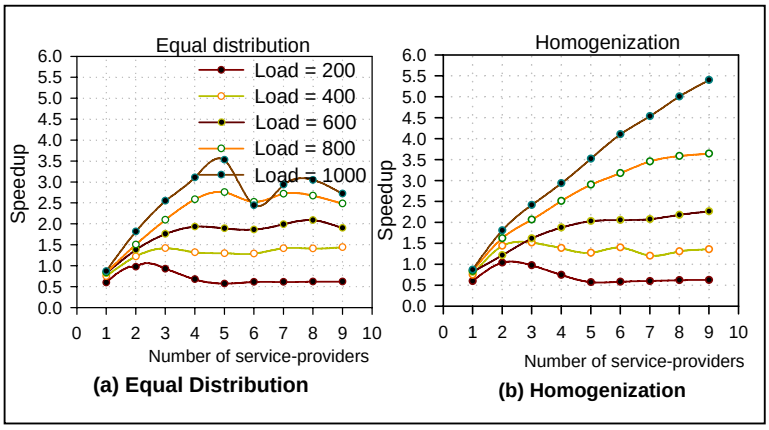

Figure 5: Speedup with different loads.

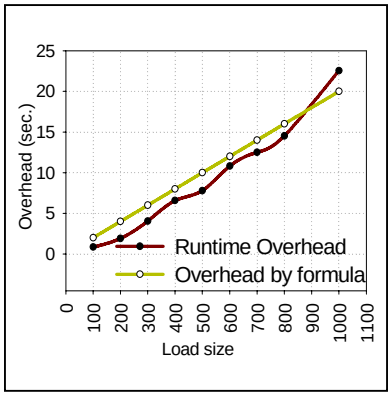

Figure 6: Overhead pattern for different load.

The corresponding homogenized performance for the same heterogeneous infrastructure is given in Figure 5 (b), which shows steady improvement of performance at higher amount of loads. Figure 5 (b) depicts that when the load size is 1000, the speedup line is almost linear, while at lower sizes speedup is more nonlinear. This proves that overhead becomes negligible for huge amount of load hence speedup becomes a linear function.

A comparison between Figure 5 (a) and Figure 5 (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 describe the nobility of homogenization through TDA.

The overhead function is a linear one. ***M***, of Eq. (9) is 20 for our network. Formulas overhead and real overhead for different load size are sketched in Figure 6.

3.2 Non-divisible Problems

Performance is measured in an existing distributed system that solves Constraint Satisfaction Problems[12] using parallel computers by a random initialization search based heuristic with the Global Evaluation Function (GEF)[14]. For an instance, N-queen problem is established. Introduction of agent based Global Evaluation Function is not the best way of the solution of N-queen problem as already some analytical solutions for the problem exist[13]. The goal of homogenization is not to find out the best algorithm for a specific problem but to harness computation power from the maximum utilization of the networked environment.

A machine that runs multiple agents concurrently by different independent threads would result in a system state that possesses the probable lower evaluation value and the likelihood to be closer to the solution. As the number of threads increases in this standalone machine, the likelihood that some of the agents are near to the solution is increased. Figure 7 shows the experimental results. Performances are taken for various numbers of queens, *N*.

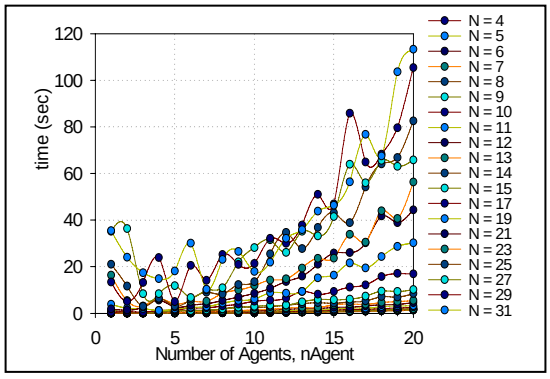

Figure 7: GEF for different number of queens (nAgent vs. time).

For each *N*, time to find the solution is taken, with different number of agents. A maximum of 20 agents are taken for each *N*. The experiment shows that it is better to use two agents rather than using one, again it is better to use three agents rather than two.

Further inference of the statement is not true i.e., after introduction of the third agent, the search time eventually increases with the involvement of further agents in the machine. At every level, requests are made several times and average time is taken to plot the graph of Figure 7. Figure 7 suggests that if the number of agents in this

machine is equal to three then the machine would derive the solution in the shortest possible time. So *maXAgent* for this machine is equal to three. This number would vary machine-to-machine depending on the performance of the host.Figure 8 depicts the behaviors at different performance levels where the number of queens is 30 in a (30X30) chessboard. Performances are taken at 0.5854, 2.7884, 5.9882, 6.9501, 7.9712 and 8.3193 hundred thousand keys/sec. *maXAgent*s are found to be 7, 7, 7, 5, 4 and 3 respectively. The runtime behavior of *maXAgent* follows Equation (10). The runtime behavior and the formula behavior are plotted in Figure **9**. Therefore, homogenization suggests *k* number of agents in a host where *k* is equal to *maXAgent* for that particular machine. So, total number of agents, *totAgent* in the system would be

$$totAgent = k_0 + k_1 + k_2 + ... + k_{m-1} \cdots\cdots\cdots (11)$$

where *m* is the number of service-providers in the system.

Homogenization provides better result even in a standalone machine that is solving non-divisible problems using TDA server. In this experiment, standalone systems of six different performances of Figure 8 got speedups of 45, 177, 4, 5, 2 and 2 respectively.

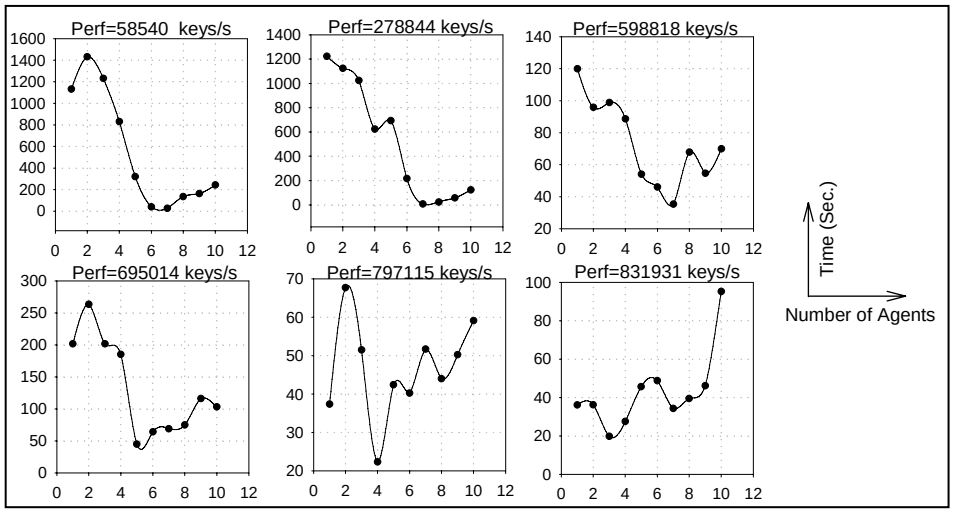

Figure 8: Detection of *maXAgent* at different performance levels.

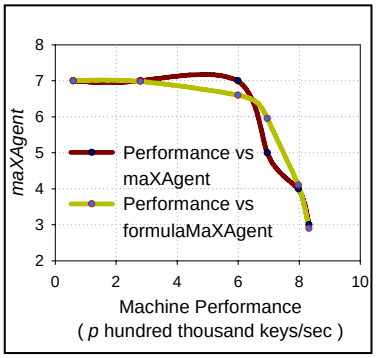

Figure 9: Homogenization pattern for *maXAgent*.

4. Conclusion

Homogenization is not limited to only linearly divisible problems. Besides, it produces better result for non-divisible independently executing parallel processes. It can be used for solving computation intensive scientific problems. Commercial applications, where millions of records are manipulated and thousands of clients are to be served, homogenization would provide better performance. Homogenization does not require any kind of user interaction for its knowledge centric distribution mechanism. When someone is exploring the space, it becomes challenging to detect which part of the space is poor and which part is rich. Homogenization enables system independent performance evaluation, which is preserved by the TDA server. As a result homogenization would become applicable in wide range of problem space. Homogenization can be used to enrich the system with

Migration of Thread [11] from the weaker part to the rich part of the space. The future direction of the work is to provide Distributed Artificial Intelligence (DAI) with the environment of Distributed Intelligent System (DIS) to secure maximum possible speedup not only homogenized by the server but also gained by intelligent service providing agents.

Homogenization technique based of TDA, is an enriching mechanism of job distribution. 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 for a linearly divisible problem. It is established with an automatic manner in TDA as a transparent load-balancing tool. In a non-divisible independent parallel system homogenization provides speedup even if there exists just a single service-providing machine. 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.

5. References

[1] A. Baratloo, M. Karaul, Z. Kedem & P. Wyckoff, "Charlotte: Metacomputing on the Web", *Future Generation Computer Systems*, *15*(5-6), 1999, 559-570.

[2] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, & D. Wu., "Javelin: Internet-Based Parallel Computing Using Java", *Concurrency: Practice and Experience*, *9*(11), 1997, 1139-1160.

[3] M. M Fuad & M. J. Oudshoorn, "AdJava - Automatic Distribution of Java Applications", *Australian Computer Science Communication*, *4*(28), 2002, 65-77.

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

[5] R. V. Nieuwpoort, T. Kielmann & E. B. Henri, "Efficient load balancing for wide-area divide-and-conquer applications", *ACM SIGPLAN Notices*, *36*(7), 2001, 34-43.

[6] M. S. Hossain, K. M. N. H Khan, M. M. Fuad & D. Deb, "Triangular Dynamic Architecture for Distributed Computing in a LAN Environment", *Proceedings of 6th International Conference on Computer and Information Technology*, *2*(1), Dhaka, Bangladesh, November 2003, 481-486.

[7] M. S. Hossain, M. M. Fuad, D. Deb, K. M. N. H Khan & M. M. A. Joarder, "Load Balancing in a Networked Environment through Homogenization", *Proceedings of 10th International Conference on Cybernetics and Information Technologies (ISAS-CITSA 2004)*, *3*(1), Orlando, Florida ~ USA, July 2004, 99-104.

[8] R. L. Hyde & B. D. Fleisch, “A Case for Virtual Distributed Objects”, *Int'l Journal on Parallel and Distributed Computing*, *1*(3), 1998, 1-11.

[9] Sun Microsystems, “The Real-Time Specification for Java”, *www.java.sun.com*, Last Visited: July 8, 2004.

[10] Sun Microsystems, “Java Remote Method Invocation Specification”, *http://java.sun.com/j2se/1.4.2/docs/guide/ rmi/spec/rmiTOC.html*, Last Visited: July 8, 2004.

[11] K. Thitikamol & P. J. Keleher , “Thread Migration, Load Balancing and Heterogeneity in Non-Dedicated Environments”, *Proceedings of the 2000 Int'l Parallel and Distributed Processing Symposium*, May 2000, 583-588.

[12] Gu J., “Local Search for Satisfiability (SAT) Problem”, *IEEE Trans. On Systems, Man and Cybernetics*, *23*(4), 1993, 1108-1129.

[13] Gu J. & Sosic R., “A Polynomial Time Algorithm for the N-Queens Problem”, *SIGART bulletin*,*1*(3), 1990, 7-11.

[14] Jing H. & Qingsheng C., “Emergence from Local Evaluation Function”, *SFI (Santa Fe Institute), WP 02-08-036*, August 2002.