

A STRATEGY FOR TUNING THE PERFORMANCE OF DISTRIBUTED SHARED MEMORY SYSTEMS

MOHAMED KHALIL and EVTIM PEYTCHEV

*School of Computing and Technology, The Nottingham Trent University,
Burton Street, Nottingham, NG1 4BU, UK
mohamed.khalil@ntu.ac.uk, evtim.peytchev@ntu.ac.uk*

Abstract: The increasing demands of distributed and parallel applications require sufficiently high-performance distributed shared memory algorithms capable of providing the service almost steadily. Past and recent decades witnessed the introduction of number of algorithms and strategies aim at tuning and optimizing the performance of distributed systems and at extending the scalability of the systems. This paper presents a new strategy that adjoins an intermediate level of control to the system to scale up the performance via relieving the overhead at the main server. This strategy can be implemented at both setup- and run-time.

Key words: DIME-II, Round-trip time, Intermediate server, optimization, shared memory, distributed computing.

INTRODUCTION

Building distributed systems on network of workstations with distributed shared memory (DSM) algorithm has been proved as a viable alternative to the traditional message-passing paradigm as the speed of the network connections and the computational power of the computers grow at steady pace. The increasing demands of distributed applications require sufficiently high-performance DSM algorithms. A research direction has been launched, alongside developing efficient DSM algorithms, to investigate new techniques for improving and enhancing the performance of DSM systems through improving the network connectivity in the system.

Such techniques can be called complementary techniques as they are used in conjunction with DSM algorithms. This direction of research has presented a wide spectrum of techniques to optimize the performance of distributed and networked systems at different levels of optimization. Levels of optimization techniques range from client interface, through middleware and servers, to the communication infrastructure. Among these techniques are: adaptive protocols that adjust with the memory access pattern in distributed applications [Amza et al, 1999]; per-node multithreading [Mueller, 1997]; relaxing consistency definitions to match the needs of an application, load sharing...etc.

Many different protocols have been proposed for implementing a software shared memory abstraction on distributed memory hardware environment. The relative performance of these protocols is application-dependent in such away that the memory access

patterns of a distributed application determine which protocols exhibit good performance. Therefore, building distributed systems with different protocols may boost the performance in the sense that the system can choose at run-time the right protocol based on the observed access patterns.

In [Amza et al, 1999] experiments were conducted to demonstrate the benefits of having protocols that automatically adapt at run-time to the memory access patterns observed in the applications on the assumption that shared memory access patterns are detected using virtual memory protection scheme in the hosting machine. The experimental reports concluded that adapting protocols at run-time according to access patterns of the shared memory can effectively improve the performance only with respect to the distributed applications, as it degrades the performance of others.

Also, many algorithms have been presented to achieve optimization via redistributing tasks between processors in order to ensure that no processor remains idle. Such algorithms called load sharing algorithms [Karatza and Hilzer, 2002]. In these algorithms, the load distribution activity is initiated in two different ways. With sender-initiated algorithms, the activity is initiated when an over-loaded node (sender) attempts to send task to another under-loaded node (receiver). On the other hand, receiver-initiated algorithms trigger the activity when an under-loaded node (receiver) requests a task from an over-loaded node (sender).

In [Karatza and Hilzer, 2001] a new epoch load sharing strategy was presented. With this policy,

workload is uniformly distributed among workstations using job migration which takes place only at the end of predefined intervals called epochs. The aim of this model is to reduce the number of times that global system information is needed to make allocation decision while obtaining good overall performance. The scheduler starts collecting information at the end of the epoch. The scheduler collects the information about the status of all workstation queues, evaluates the mean of all queue lengths and places processor queue length into increasing order in a table. After collecting the information, jobs are transferred from the most heavily loaded processors to the lightly loaded ones. This process continues until either all processors have queue lengths equal to the mean or some of them differ at most by one job.

To evaluate its performance, the epoch algorithm was executed with different epoch sizes and compared with some of the other similar models, such as the migratory probabilistic and shortest queue models [Karatza and Hilzer, 2001]. The comparison concluded that for all level of migration overhead with different workloads, epoch model with different epoch sizes involved much less overhead, in terms of collecting global system information, than the shortest queue policy and the migratory probabilistic method. Also, the performance of epoch strategy with small epoch sizes performed comparably to the performance of shortest queue method.

This paper introduces new strategy for enhancing the performance of DSM systems via maximizing the data retrieval rate at application level at run-time. Generally speaking, this algorithm enhances the performance of DSM systems by adding up intermediate level of control to support the main server supplying the service to the currently running applications. This paper uses DIME-II DSM system to demonstrate the algorithm.

DIME-II SYSTEM

DIME-II [Khalil and Petchev, 2003] is a distributed shared memory system that adopts a framework of two levels of storage space [figure 1]. The higher level of storage, called the original memory, contains the original copies of the shared memory of the system and controlled by the main server, called DIME-II server. The lower level of storage space, called the intermediate memory, keeps copies of part of the original memory needed by certain application.

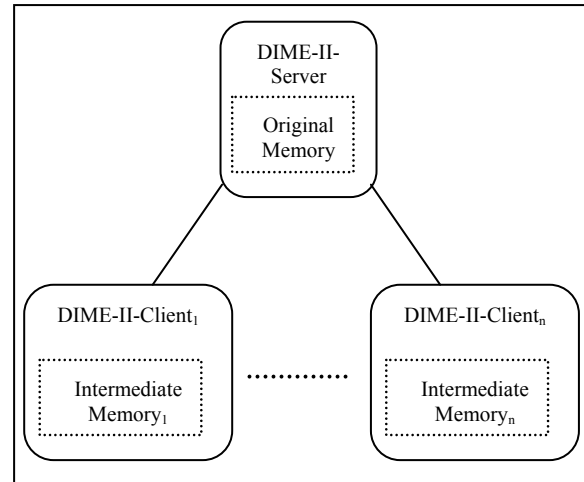


Figure 1: DIME-II's Levels of Storage Space

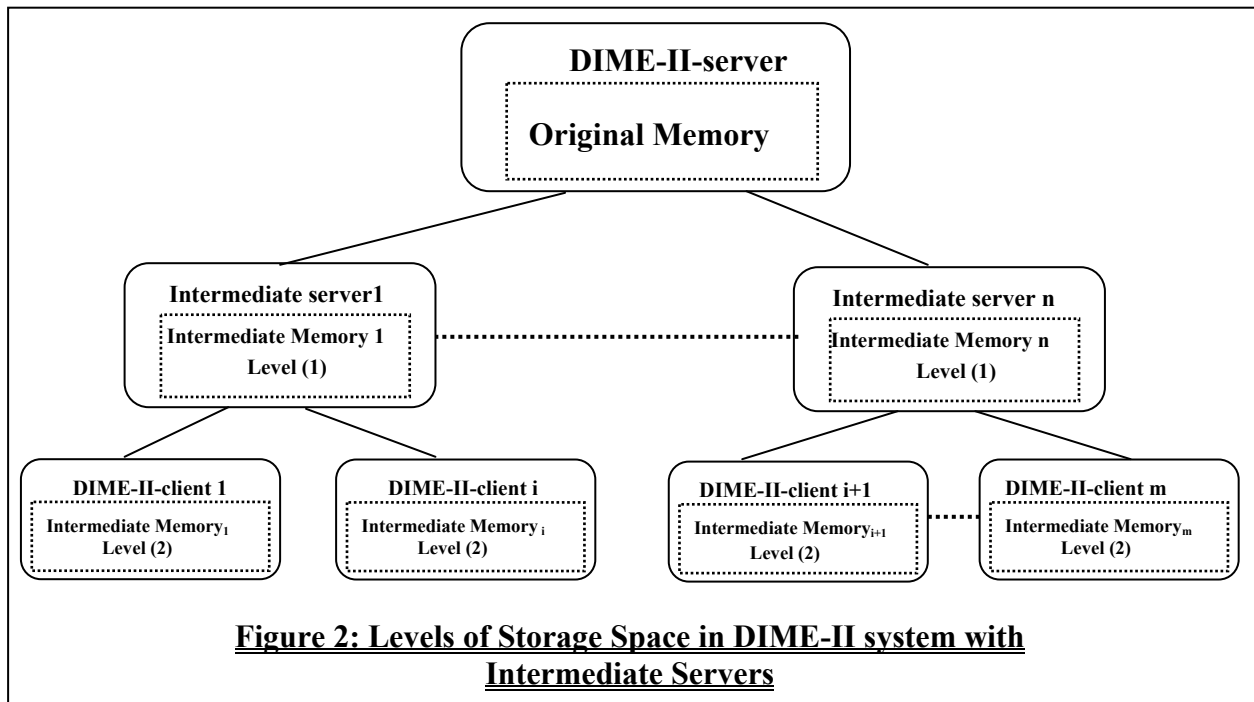
This intermediate memory resides in the machine where that specific application is running and controlled by a subsystem called DIME-II client. DIME-II-client acts as a server at this level of control as it keeps the intermediate memory consistent with the original memory, and at the same time it provides service to its application.

ROUND-TRIP TIME-BASED ALGORITHM

This algorithm aims at scaling up the performance of DIME-II DSM systems via adding intermediate level of control to the system. The sought goal is to alleviate the overhead on the main central server, by initiating another level of service that can supply the running application with the service rather than taking it directly from the main server. Having several levels of control can allow more applications to be added to the system while maintaining good performance (i.e. good scalability).

The extra level of control or service consists of several intermediate servers and memories. Each intermediate server supplies the service to certain amount of applications, and therefore, the memory associated with it can contain only the data needed by these applications, which means this lower level of memory structure contains only part of the original memory.

The location of this server lies between DIME-II server and DIME-II clients (figure 2). That gives three levels of memory structure in DIME-II of multi-level storage spaces. 1. The main (original) memory, which is controlled by DIME-II-server. 2. Intermediate memories (level 1) controlled by intermediate servers. 3. Intermediate memories (level 2), which are under the control of DIME-II-clients.



To implement this strategy, a crucial decision has to be taken carefully of when to initiate this level and where to place the new intermediate server and memory. Such level of control can be initiated either at setup-time or run-time.

STATICALLY-INITIATED INTERMEDIATE SERVERS (SIS)

As it aims to improve the performance of distributed shared memory system by adding another level of control reducing the overload occurs at the main server, an intermediate server and memory can be placed in a location nearby specific applications. For instance, if the main server resides in a far-off location, remote applications can have intermediate server in the same LAN. Therefore, remote applications can get the service of DIME-II DSM system via an intermediate server associated with storage space. The intermediate server communicates with the main server on behalf of the remote applications. Therefore, the main server gives the service to only one application (i.e. the intermediate server) and therefore it will have minimum overhead. Such servers are called statically-initiated intermediate servers (SIS), because they are initiated at setup time rather than run time.

To examine the impact of having intermediate level of control on the performance of DIME-II system, experiments have been launched to compare the performance of the system with and without intermediate servers. For these experiments, applications that read and write continuously on the

DIME-II system are used. Those applications are placed on the same LAN with an intermediate server, whereas DIME-II server is located on a location connected with the applications' LAN by five switches. The experiments have been executed with different workloads (Table 1). Each workload executes only one writer and different number of readers. Here, workload 1 means one reader and one writer, and workload 2 means two readers and one writer, and so on.

Workload	Without SIS	With SIS
Workload1	134	124
Workload2	112	121
Workload3	80	118
Workload4	71	102
Workload5	58	89

Table 1: The performances of DIME-II with and without static intermediate server (SIS) measured as data retrieval rates (kilobytes/second)

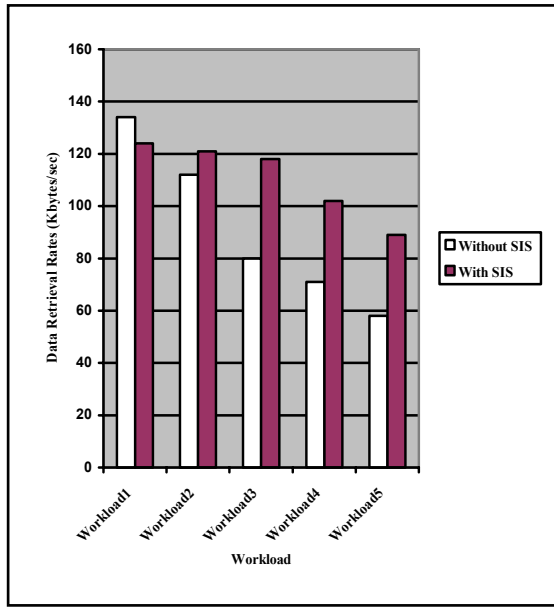


Figure 3: The performance of DIME-II system with and without Statically-initiated Intermediates Servers

The results show that, having static intermediate servers does not improve the performance when there are only two applications in the system (Workload 1). However, the impact of the presence of the intermediate control level becomes visible as the number of application increases (Figure 3).

Thus, statically-initiated intermediate servers can significantly improve the performance of DSM system even when the number of applications is not large. Such static servers can be used more effectively when having, for example, DSM system that is distributed in a very wide area (i.e. different cities or may be countries).

Therefore, such servers can be used when the main server resides in a far-off location, which allows remote applications can have intermediate server located in the same LAN where they can take the service. Experiments have shown that such servers can improve the performance by up to 50%.

DYNAMICALLY-INITIATED INTERMEDIATE SERVERS (DIS)

The encouraging results from the implementation of SIS within DIME-II system can be taken one step further to examine the initiation of intermediate servers at run time. This section presents an algorithm that can initiate intermediate servers at run time, when the system has three applications (only for demonstration purposes).

When DIME-II server is supplying the service to three applications, it can direct any further connection request from new application to get the service from the nearest intermediate server. In this algorithm, each DIME-II client in the system has an embedded intermediate server that is ready to provide the service to the requesting applications. DIME-II server keeps records of the available intermediate servers. DIME-II server gets the IP address and the port number where an embedded server is listening, when it receives a connection request from the application where an intermediate server is embedded. Hence, in the absence of any static intermediate servers, there will be intermediate server of number equal to the number of the applications that get the service from the main server, DIME-II server.

When it receives a connection request from new application, DIME-II server sends a message to that application to get the service from an available intermediate server. This message contains the IP address and port numbers of all intermediate servers in the system. The application, via its DIME-II client, sends dummy messages to the intermediate servers to calculate the round-trip time (RTT) between the application and each of the servers. Assuming that the server of the shortest RTT is the nearest one, the application can take the service from the intermediate server of the shortest RTT. This is based on the results of SIS servers, that the performance is much better when an application gets the service from a nearby intermediate server. Such servers are called dynamic intermediate servers (DIS) as they are initiated at run time. To examine the effectiveness of the algorithm, the same benchmark in the previous experiments is used. This time, workload 1 contains 3 applications and 4 applications in workload 2, and so on (Table 2).

Workload	Without DIS	With DIS
Workload1	87	104
Workload2	70	99
Workload3	57	85
Workload4	50	74

Table 2: The performances of DIME-II with and without dynamic intermediate server (DIS) measured as data retrieval rates (kilobytes/second)

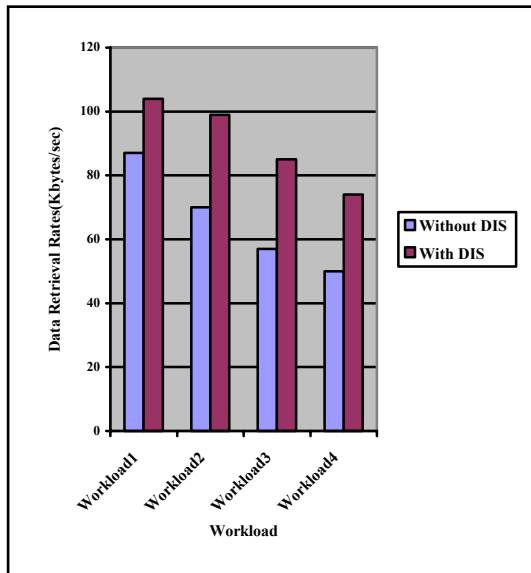


Figure 4: The performances of DIME-II with and without dynamic intermediate server (DIS)

Experimental results have shown that the algorithm works fine every time a new application is requesting the service, and the performance can be improved by up to 48%. Therefore, initiating intermediate servers that are embedded within DIME-II client code is a practical means to produce an improved performance in DIME-II DSM system, when DIME-II server is loaded with applications. Furthermore, round trip times can be used to identify the location of the intermediate server, where an application is preferred to get the service from.

CONCLUSION AND FUTURE WORK

This paper presents a new strategy for scaling up the performance of distributed shared memory systems. This strategy adds intermediate level of control to alleviate the overhead at the server side of the system. This level of control can be initiated at setup time or run time. Experiments have shown that having an intermediate level of control in DSM system can improve the performance. The performance of the system is increased by up to 50%, measuring the performance in terms of data retrieval rates.

This algorithm can be extended to allow the system to reconfigure its own structural design by adding number of intermediate servers and redirecting the connected applications to get the service from the available server according to the round-trip times, to adapt to the current state of the system. Although it is always difficult to reconfigure DSM systems at run time to scale up the system, as it usually involves

high software and communication overhead. However, having such algorithm is always an attractive solution, as the system can be monitored all the time and reconfigured when certain overhead is reached. Thus, it provides the using system a better scalability and, of course, an optimized performance. The main reason for improvement in the system is the fact that the algorithm cuts the network latencies in the data retrieval process.

REFERENCES

Amza C., Cox AL, Dwarkadas S, Li-Jie J., Rajamani K, Zwaenepoel W., "Adaptive protocols for software distributed shared memory", Proceedings of the IEEE, vol.87, no.3, March 1999, pp.467-75. Publisher: IEEE, USA.

Mueller F., "Distributed Shared-Memory Threads: DSM-Threads", Workshop on Run-Time systems for Parallel Programming, Apr 1997.

Khalil M., Peytchev E., "Traffic Telematic Computing Framework based on Non-Locking and Housekeeping Distributed Shared Memory Algorithm", Sixth United Kingdom Simulation Society Conference (UKSIM2003), Apr. 2003, Emmanuel college, Cambridge, UK.

Karatza H., Hilzer R., "Epoch Load Sharing in a Network of Workstations", in the Proceeding of the 34th Annual Simulation Symposium (SS '01), Seattle, WA, April 2001.

Karatza H., Hilzer R., "Load Sharing in Heterogeneous Distributed Systems", in the Proceeding of the 2002 Winter Simulation Conference, San Diego, California, December 2002.

BIOGRAPHIES



Mr. Mohamed Khalil is a Research Student at the School of Computing and Technology, the Nottingham Trent University. He was graduated from Faculty of Mathematical Sciences, University of Khartoum, Sudan with a bachelor degree (honor) in computer sciences. He worked in the Department of Computer Sciences, Faculty of Mathematical Sciences, University of Khartoum as a teaching assistant for nearly three years. He won two university prizes for the best academic performance while he was student in the academic years 93/1994 and 96/1997. Mr. Khalil started his PhD in August 2001 and the title of his thesis is "Integrative

Monitoring and Control Framework Based on Software Distributed Shared Memory Non-Locking Model” under the supervision of Dr. Evtim Peytchev and Prof. Andrzej Bargiela. His research interests are: Distributed shared memory algorithms and prototyping and optimization, and Traffic Telematics Systems.



Dr. Evtim Peytchev is a Senior Lecturer at the School of Computing and Mathematics, the Nottingham Trent University and has been a member of Intelligent Simulation and Modelling group for 11 years. Most of the recent research work in the group, dealing with the traffic control telematics, has been carried out by Dr. Peytchev under the supervision and leadership of the head of the RTTS group Prof. Andrzej Bargiela. As a result of the research work Dr. E. Peytchev has successfully presented his Ph.D. work entitled “Integrative Framework for Discrete Systems Simulation and Monitoring”. He worked as a researcher for the successful conclusion of an EPSRC project “Integrative framework for the predictive evaluation of traffic control strategies” (GR/K16593) and most of his publications reflect the work under this project. Dr. Peytchev’s interests span: traffics simulation modelling, traffic Telematics, mathematical modelling of the uncertainties in traffic, distributed computing environments, shared memory design, Telematics technology application in the urban traffic control. He is involved in International collaboration with the Transportation Systems Laboratory at the Helsinki University of Technology (Dr. I. Kossonen) and he is principal investigator for Nottingham Trent University in the DTI funded ‘Traffimatics’ project.