DIME-II: A COMPUTING FRAMEWORK FOR TRAFFIC SYSTEMS

MOHAMED KHALIL and EVTIM PEYTCHEV

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

Abstract: Building a successful distributed shared memory system depends enormously on the degree of consideration of certain design issues in the designing stage. This degree varies according to the nature of the distributed application itself. DIME-II, an extension of DIME-I system, is designed with features specific for traffic control distributed systems taken into account. The paper presents implementation of this system considering number of common design issues and inheriting some features from the implementation of DIME-I.

Key words: DIME, Granularity, Scalability, Heterogeneity, Distributed Computing.

1. INTRODUCTION

Over the last years, distributed shared memory (DSM) paradigm has attracted researchers who have investigated different approaches that hide remote communication mechanism from the programmers on a cluster of workstations, where each workstation has computing power comparable to the mini-mainframe in the past.

Many of Distributed shared memory (DSM) algorithms have been successfully implemented in a wide range of experimental and commercial applications. Building an efficient, successful software distributed shared memory system depends mostly on the application that implements the DSM algorithm. However, there are number of requirements or designing issues which influence the performance and the efficiency of the system, as presented in [Nitzberg B. et al, 1991]. The level of satisfying these issues varies from one application to another. Therefore, considering the nature of an application in the designing stage can effectively increase the performance of that application. These design issues are: structure and granularity; scalability; heterogeneity; and memory consistency.

The **DI**stributed **M**emory **E**nvironment (DIME) [Argile A. et al, 1999] is a software DSM system that provides an interface between distributed software modules that execute on networked workstations. DIME has been designed specifically to support vast range of transport telematics applications and it offers a convenient interface to the applications programmer. The first implementation of DIME system is called DIME-I. As it was built as a user-level software DSM system, DIME-I provides an easy to use communication interface that simply and reliably delivers data and messages to all nodes in the system. In [Khalil M. et al, 2003] a revised framework of DIME-I was introduced in order to improve the performance of DIME system mainly by avoiding its limitations and minimizing the time of data retrieval from the viewpoint of user application. This new framework is called DIME-II. This paper presents an implementation for DIME-II.

The presented implementation chooses to continue using user-level implementation for DIME-II software DSM system, as it does not require changes in the lower levels of the system (compiler and operating system). Besides, such implementation provides good portability in distributed systems. There are some pre-existing requirements that have been taken into account in the designing stage of DIME-II. For example, the implementation of DIME-II, as in DIME-I, supports two types of data structures that naturally exist in urban traffic information and control system. Other inherited properties from the DIME-I system are the granularity and the non-locking approach.

2. TYPES OF DATA IN TRAFFIC CONTROL SYSTEM

Building a successful DSM system requires a detailed knowledge of the data transactions in the system; therefore a special consideration of data flows in a traffic control system was taken in the design stage of DIME-I [Peytchev E., 1999]. In a traffic control system there are two kinds of data can be recognized:

- Dynamic data: It is collected by the real-time traffic control system. It contains all information about traffic counts and local controls as they occur in the traffic network. It is characterized by its high volume - in excess of 120 Mbytes per day per one specific type of message. Besides, this

 ISBN 3-936150-25-7

kind of data is updated in a high frequency rate (per second basis).

- Static data: This kind of data is updated in a much longer period of time and its purpose (in general) is to make the results from traffic modules available for reading by the other functional modules in the system.

3. DESIGNING ISSUES

In the designing stage of building DIME-II numbers of design issues have been considered. In the traffic system there are two types of data structures: dynamic and static data. The data representing the dynamic data type is in fact a constant flow of uniform messages issued from the traffic system. To accommodate this type of data, the system needs a formation capable of accepting a number of uniform structures at a time, and at the same time keeping the most recent data only. The implementation of DIME-I utilizes circular buffer for this type of data, where each element of the buffer is a user defined message structure and its size depends on the size of the urban traffic network. On the other hand, the relatively static data in the traffic system usually reflects the value of some internal variables in the traffic modules. The volume and format of this data is usually module dependent since each module has its own internal representation of the traffic. Therefore, DIME-I utilizes an array of bytes of user's defined size for static data, as it's the most suitable choice [Peytchev E., 1999]. The implementation presented in this paper continues using this granularity, since it is suitable and convenient for representing the two types of data of traffic control systems.

Moreover, DIME-II is designed to run on different platforms, and therefore, it can run in a heterogeneous environment. However, DIME-II employs communication algorithm that internally exchanges data and messages as bytes; therefore, software modules have to make their own internal simple conversions. DIME-II is designed with capability of extension to contain further addition of software modules. As described in [Nitzberg B. et al, 1991] there are two factors that can greatly limit the scalability of distributed shared memory systems. Theses factors are: general common knowledge and central bottleneck. In DIME-II these factors have been overcome. The framework of DIME-II supports the presence of data replicas at intermediate memories each in a location near to certain application. Since each application in DIME-II performs its operation on an intermediate memory with no competition with other applications [Khalil M. et al, 2003], therefore; this housekeeping algorithm can reduce the contention on the central shared memory. Consequently, it can decrease the likelihood of central bottleneck.

4. DIME-II STRUCTURAL DESIGN

As introduced in [Khalil M. et al, 2003], the architecture of DIME-II system employs non-locking approach and consists of three layers as depicted in Figure 1. In the first layer DIME-server takes control over the original shared memory system, and has the role of monitoring any modification in the central memory in order to keep all intermediate memories throughout the system informed with the updates. In the second layer DIME-client controls accesses to certain intermediate memory that is associated with only one user application. An intermediate memory holds copy of part of the original shared memory which is required by the associated traffic module.

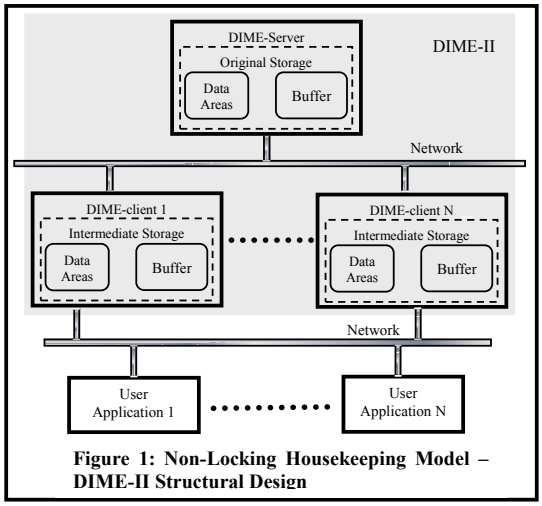

Figure 1: Non-Locking Housekeeping Model – DIME-II Structural Design

DIME-client's task is to communicate with DIME-server on behalf of its user application to perform write operations, and at the same time it looks up in the intermediate memory to retrieve certain data for the user application. In the third layer there are user applications, which are traffic control system modules. A user application performs its operations on the local memory, leaving the time delay burden of contacting the server to DIME-client for making the intermediate memory up to date, and reflecting the update in the original memory. This saves valuable time for user applications - usually wasted in network communications- to perform its native tasks.

Since this model supports the presence of data replicas, special care has been taken to avoid data inconsistency in this architecture. The consistency model presented in [Khalil M. et al, 2003] intends to maintain systemwide consistent view of the memory in terms of data area and buffer structures. This model is designed specifically to support the

two types of data structures that comprise the shared memory, and it has a flavour of sequential consistency model [Lamport L., 1979] as it is the most intuitive definition for programmers. In other words, it's a relaxed definition of sequential consistency model. Unlike sequential consistency, the consistency definition is advantageous in such a way that it supports multi-reading/multi-writing.

5. AN IMPLEMENTATION FOR DIME-II

The first decision has to be taken in this stage is where to locate the process of DIME-client in the framework. This implementation chooses to place DIME-client process at the same machine as its associated user application in order to achieve the sought goals. Therefore, slight modification to the framework in figure 1 has to be made and is illustrated in figure 2.

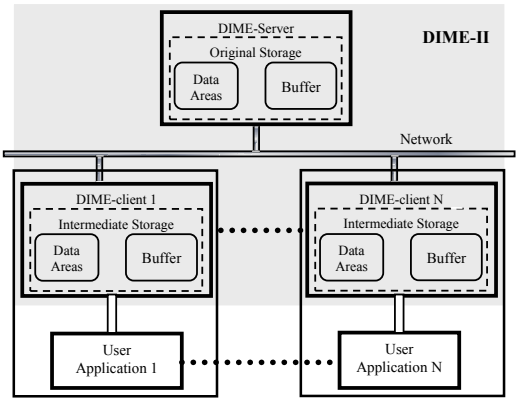

Figure 2: The current implementation of DIME-II

For implementing this framework, two separate executables have been coded for DIME-server and DIME-client. Both make use of the Java programming language.

Java's multithreaded support is essential for the successful programming of the DIME-II software. The presented implementation exploits the potential of multithreading as it has shown improved performance in DSM systems by hiding the long communication latencies typically associated with software DSM systems [Speight E. et. al, 1997] [Mueller F., 1997].

The produced software is described comprehensively in the following subsections in terms of DIME-server and DIME-client.

5.1. DIME-server

DIME-server executes command packets (cmmnd_pckt) in the order they are received (not the order they were sent). In accordance with the atomicity of DIME-II system, each command is performed as an indivisible operation. In other words, there is no interleaving when DIME-server is performing a command.

DIME-server keeps a list of every created shared item and a list for each item in the list, which contains names of user applications that currently have replica of it. When it receives a request for creating shared item, DIME-server allocates space for the item in the DSM only if it has not been created before. Otherwise, the name of the requesting application is just added to the list of the applications that have the replica of that shared item. In the case of write operations; it sends updates only to the applications that have replica of the updated value using the relative list of applications. In other words, unlike BDSM [Auld P. et al, 2000], DIME-II system employs a multicast-based algorithm to disseminate updates to the application that are involved in the write operation. On the other hand, when it receives a request for the deletion of a shared item, DIME-server deletes the name of the application from the list of that item. The item is removed permanently only if the requesting application is the last one in the list.

In order to improve the performance of DIME-server, numbers of threads are used. Each application is serviced by separate thread that listens to its requests and inserts them in a queue of command packets. This thread is called *client-service* and is created when DIME-server receives request from a user application to use the shared memory. The thread of client-service consists of two other threads: 1.*ListenToPacket* that continuously listens to command packets sent from its user application and inserts them in a queue to be processed later, 2.*SendPacket* thread that keeps checking another queue of command packets ready to be sent to the user application. All client-service threads -particularly ListenToPacket threads- insert every received command packet in one single queue. This queue is processed by another thread called *sequencer*. The sequencer is the only thread that can perform operations on the shared memory in DIME-server. After processing an operation, the sequencer passes an appropriate command packet to certain client-services, which in turn send the command to certain applications – particularly done by SendPacket thread.

Employing several threads allows dividing the task of the DIME-server into a number of sub-tasks to be executed at the same time, enhancing the functionality of the DIME-server.

The speed of the sequencer performing commands in the shared memory (read & write) is much higher than the speed of the underlying network and therefore this is not a cause for bottleneck problems. Figure 3 illustrates a general view of DIME-server.

In addition to the main task of controlling the shared memory, DIME-server holds number of permission tables. These tables prescribe levels of access that each user application has on the shared memory. The level of access is either no access, read only or read/write. DIME-server disseminates certain permission table to a DIME-client upon initiating a process of traffic module. This permission table is used by DIME-client upon performing any operation on the shared memory.

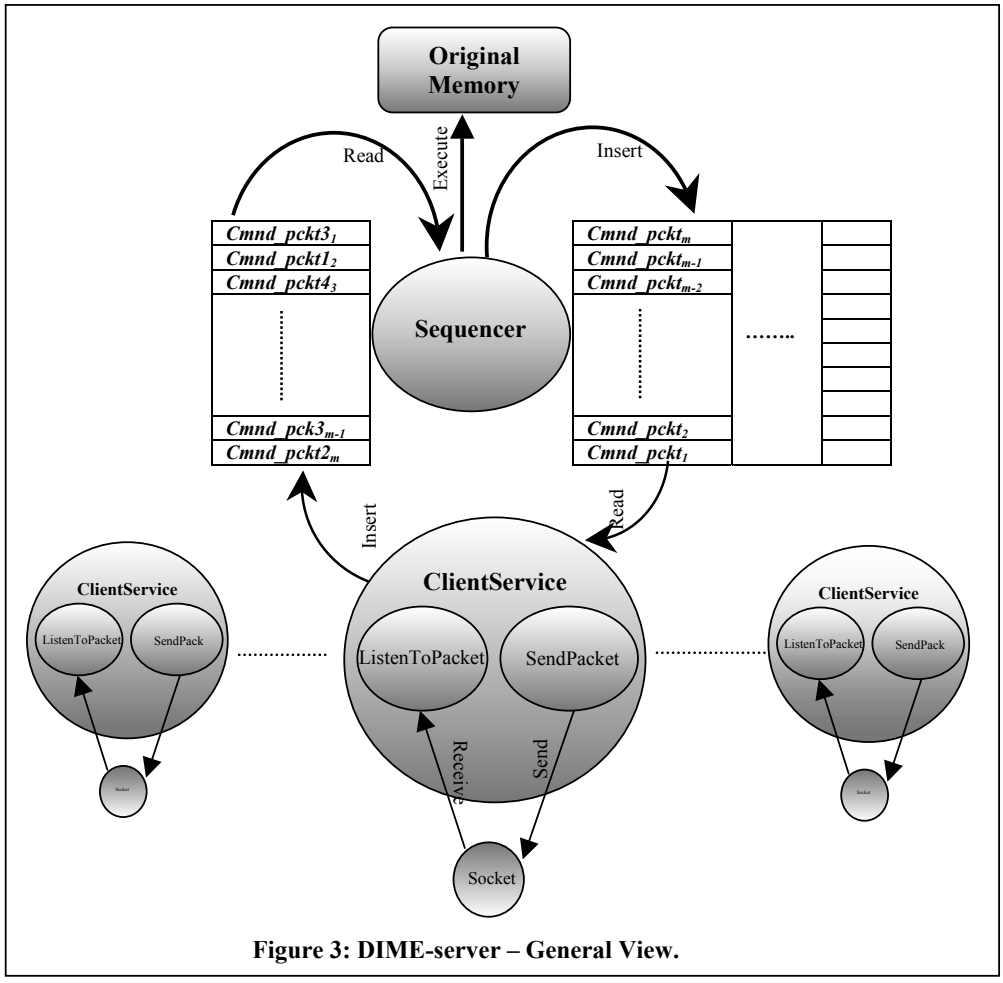

Figure 3: DIME-server – General View.

5.2. DIME-client

DIME-client keeps a copy of the permission table of its user application. It checks the privilege of its user application upon performing any operation in the intermediate memory. An operation is processed only if the application is permitted to do so, otherwise an error message is sent to it. For write operation, DIME-client can apply the update locally in the case of area writing, and then the update is sent to the DIME-server.

At DIME-client side, there is a thread that continually listens to messages from DIME-server and acknowledges them. The main task of this thread is to receive new updates from the DIME-server and then update the intermediate memory accordingly. Thereby, DIME-client can guarantee the consistency of the local replicas of the shared memory. On the other hand, any read operation can be performed directly on the available local memory without need to contact the main shared memory controlled by the DIME-server over the network. This locality of reference is advantageous in saving network bandwidth, and reducing time of data retrieval for user application.

5.3. User's Interface of DIME-II Software

So far, the produced system provides number of

functions for performing different operations on the distributed computers shared memory. These functions are:

- Initializing intermediate shared memory. User application calls this function to get permission for initializing the shared memory and start using the DSM. If the application has permission to use the system, a permit will be sent along with permission table. This permission table contains names of shared items the application is permitted to use, and the access privileges for each item.
- Creating data area/buffer. This function is called to create new shared memory item. A shared memory item is created only if the creating user application is permitted to use it.
- Writing in area/buffer. A user application invokes this function to update an existed shared item. This operation is performed only if the user application has enough permission to write in that item.
- Reading from area/buffer. The required data are sent to the requesting application along with the number of the values. Zero is sent if the requested item is empty. This operation is performed locally.
- Destroying area/buffer. Performing such operation results in removing certain shared item from the intermediate memory of the requesting application.

6. RELATED WORKS

TreadMarks [Amza C. et al, 1996] is a software DSM system where messages and data traffic is reduced by relaxing consistency semantics of the shared memory. TreadMarks is a user-level implementation of DSM relies on UNIX standard libraries in order to accomplish remote process communication, and memory management, therefore no need to make modifications on the operating system kernel. In [Lu H. et al, 1995] experimental results have shown that the separation of synchronization and data transfer and the request-response nature of data communication are responsible for lower performance comparing with PVM message-passing model. In DIME-II, there is no need for synchronization mechanism for a user application to have an exclusive access to the shared memory, since each user application is associated with an intermediate memory where all its operations are performed.

BDSM [Auld P. et al, 2000] is a broadcast-based, fully replicated software distributed shared memory system. Similar to our framework, each user process has an associated DSM subsystem that manages the shared memory, however, each user process has a complete copy of the shared memory where it processes all reads and writes locally. Also, unlike our system, all writes to memory modify the local copy and arrange to broadcast the updated values to all the other processes. Another major difference with the presented framework is that BDSM allows one user process to be executed on a workstation.

Brazos system [Speight E. et al, 1997] utilizes multithreading at both the user level and system level. Multiple user-level threads allow applications to take advantage of symmetric multiprocessor servers by using all available processors for computation. In the runtime system there are two main threads. One thread is responsible for quickly responding to asynchronous requests data from other processes and runs at the highest possible priority. The other thread handles replies to requests previously sent by the process. This multithreaded aspect of Brazos allows greater amount of computation to communication overlap. The use of separate thread to handle incoming replies allows the system to maintain multiple simultaneous outstanding network requests, which can significantly improve performance. Additionally important, the exploitation of multithreaded DSM algorithms proved significant in hiding the communication latencies [Muller F., 1997].

7. CONCLUSION

This paper presents description of an implementation of the framework of DIME-II. This implementation is believed to improve the performance of the whole system in many aspects: saving network resources, reducing data retrieval from user application viewpoint, performing number of tasks per-node simultaneously, and at the same time maintaining consistency by a simple straightforward model. Currently number of experiments are set-up and under way in order to evaluate the performance of the DIME-II system in comparison with the current DIME-I system

8. REFERENCES

[1] Argile A., Peytchev E., Bargiela A., Kossonen I., “Dime: A Shared Memory Environment for Distributed Simulation, Monitoring and Control Of Urban Traffic”, 8th European Simulation Symposium, Genoa, Italy, ISBN 1-565555-099-4, Vol.1, pp. 152-156.

[2] Peytchev E., “Integrative Framework for Discrete Systems Simulation and Monitoring”, Ph.D. thesis, Department of Computing, The Nottingham Trent University, Nottingham, England. Feb. 1999.

[3] Nitzberg B., Lo V., “Distributed shared memory: A survey of issues and algorithms”, IEEE Computer, vol. 24, pp. 52--60, Aug. 1991.

[4] Khalil M., Peytchev E., "Traffic Telematic Computing Framework based on Non-Locking and Housekeeping Distributed Shared Memory Algorithm", Sixth United Kingdom Simulation Society Conference (UKSim 2003), Apr. 2003, Emmanuel college, Cambridge, UK.
[5] Lamport L., "How to make a Multiprocessor Computer that correctly executes Multiprocessor Programs". IEEE Trans. Comp., vol. C-29, no. 9, pp. 690-691, Sept. 1979.
[6] Auld P., Kearns P., "Broadcast Distributed Shared Memory", Proceedings of the ICSA 13th International Conference on Parallel and Distributed Computing Systems, ICSA, pp., 2000.
[7] Speight E., Bennett J., "Brazos: A Third Generation DSM System", In Proceedings of the 1st USENIX Windows NT Symposium, pp. 95-106, August 1997.
[8] Mueller F., "Distributed Shared-Memory Threads: DSM-Threads", Workshop on Run-Time systems for Parallel Programming, Apr 1997.
[9] Amza C., Cox A., Dwarkadas S., Keleher P., Honghui L., Rajamony R., Weimin Y., Zwaenepoel W., "TreadMarks: Shared Memory Computing on Networks of Workstations". Computer, vol.29, no.2, Feb. 1996, pp.18-28. Publisher: IEEE Comput. Soc, USA.
[10] Lu H., Dwarkadas S, Cox AL, Zwaenepoel W. "Message Passing versus Distributed Shared Memory on Networks of Workstations". Proceedings of the 1995 ACM/IEEE Supercomputing Conference (IEEE Cat. No.95CB35990). ACM. Part vol.1, 1995, pp.865-906 vol.1. New York, NY, USA.

Mr. Mohamed Khalil is a Research Student at the School of Computing and Mathematics, the Nottingham Trent University. He was graduated from Faculty of Mathematical Sciences, University of Khartoum, Sudan with a bachelor degree (honour) 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, Distributed Shared memory Applications, 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 10 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. Kosonen) and in the DTI funded 'Traffimatics' project.