Performance Comparison of Process Migration with Remote Process Creation Mechanisms in RHODOS

D. De Paoli, A. Goscinski, M. Hobbs and P. Joyce
School of Computing and Mathematics
Deakin University, Geelong
Victoria 3217, Australia

{ddp, ang, mick, philj}@deakin.edu.au

Abstract

We claim in this paper that both remote process creation and process migration are efficient mechanisms to be used in the improvement or development of high performance computer systems. In particular, we demonstrate that the claims made by some researchers that process migration is too heavy to be used to support dynamic load balancing are unsubstantiated. We support our claim by presenting these two mechanisms available in the RHODOS distributed operating system, comparing and contrasting these mechanisms and reporting on their performance.

1 Introduction

Modern computer environments are increasingly being based upon groups of workstation and/or personal computers interconnected via high speed local and wide area networks. This is primarily due to the fact that the performance per cost ratio is extremely high and this ratio is increasing. The environment for a network of workstations has the ability to be a distributed system when resources such as file systems, printers, special devices and even processors are transparently shared throughout the network. This requires the employment of a distributed operating system [1]. The first benefit of such a system is that it allows the balancing of the load over all workstations, which is desirable for several reasons: an improved overall performance and a better utilisation of resources, which leads to a improved cost effectiveness. Another benefit relates to an emerging trend in the use of distributed systems. This trend is moving towards the utilisation of a distributed system in the provision of distributed parallel environments, which allows the execution of processes in parallel as well as traditional concurrent execution. This area is increasing in popularity since the overall processing capacity of workstation networks are becoming comparable with that of super computers [2].

Static allocation has been proposed as a feasible approach to be used to balance load and to improve the overall performance of distributed systems. This move has been strongly supported by the application of the PVM package to execute applications in parallel on a network of workstations [3]. This implies that making a decision on where to create and run a new process could be done efficiently and that process creation on a remote workstation is inexpensive.

Dynamic load balancing has been recognised as a potential candidate to balance load. However, some researchers [4] claimed that dynamic load balancing cannot be used in real systems as process migration, a mechanism proposed to be used to support load balancing, is not fast enough.

Our initial research into process migration [5] indicates that process migration can be effectively used in real systems to support dynamic load balancing. This line of thought has also been presented by A. Barak and his group, who demonstrated just recently [6] that load balancing supported by process migration can improve performance of PVM on a network of workstations.

However, there has been no published work which presented a performance comparison of process migration and remote process creation mechanisms on real network of workstations/distributed systems, supported by one distributed operating system. Therefore the goal of this work is to present these two mechanisms developed as a part of the ResearcH Oriented Distributed Operating Systems (RHODOS). We also report on our experience with these two mechanisms, the performance of them and address the issue of their applicability in a real system in order to create a high performance computer system.

2 Global scheduling and RHODOS

This section reports on both Global Scheduling as a way to provide the support for high performance computer systems and the RHODOS system.

2.1 Global scheduling enabling high performance systems

A considerable amount of effort has been devoted to the research of both Static Load Balancing known also as Static Allocation and Dynamic Load Balancing algorithms and methods. Briefly, Static Allocation is the placement of a process on an underloaded workstation at the time of the process' inception; while Dynamic Load Balancing is the

This work was partly supported by the Australian Research Council under Grant A49232429 and the Deakin University Research Grant 05040110151.

movement of an already executing process (or processes) from overloaded to underloaded workstations. To date research of these topics have been separated, either focusing exclusively on Static Allocation methods or exclusively on Dynamic Load Balancing. The feeling of researchers in these areas are that both methods were incompatible. Each method performs well under certain load conditions and these conditions have been used as arguments why the other method was not viable. For example, Static Load Balancing works well when the load of a workstation is continually high with little fluctuations, thus all new work (in the form of new processes) are then created remotely without making the load of the local workstation worse. Conversely, if the load fluctuates greatly from lightly loaded to heavily loaded then Dynamic Load Balancing is an appropriate method to move excess load from an overloaded workstation to a lightly loaded workstation.

One example proposed against static allocation is that user autonomy cannot be guaranteed. An idle workstation may be involved in static allocation, but when the owner returns and logs into that workstation, it is overloaded with foreign processes and has no way of rectifying the situation. On the other hand, dynamic load balancing is said to be too complicated and the overheads incurred are too great to achieve the required rate of load balancing, particularly when the number of workstations involved in the dynamic load balancing interaction increases. Since a considerable amount of information must be shared and distributed between all participants and running processes must be migrated.

We propose to employ Global Scheduling which is the combination of two approaches: Static Allocation and Dynamic Load Balancing, implemented with the objective of achieving an overall balanced load. To support static allocation and dynamic load balancing a number of mechanisms are required. Static allocation requires initial placement which can be provided by a remote process creation service, whilst dynamic load balancing requires a process migration service. Location transparency involves process relationship (parent and children running on different workstations), communication and resource access transparency. Modern distributed operating systems provide the required level of location transparency [1], [5]. The final support required by global scheduling is the collection mechanism.

2.2 Proposed global scheduling structure

Separated into two distinct regions, the global scheduling facility proposed in Figure 1 is a combination of policy and mechanism components.

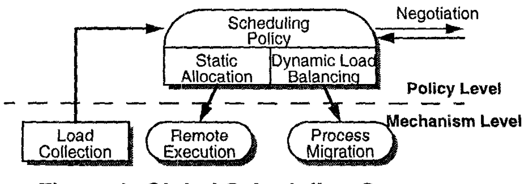

Figure 1: Global Scheduling Components

The scheduling and decision making algorithms are located within the scheduling policy component. The scheduling policy uses data presented by the load collection mechanism to switch between the Static Allocation and Dynamic Load Balancing strategies. Where the Static Allocation facility utilises the remote process creation mechanism and the Dynamic Load Balancing facility utilises the process migration mechanism.

There is the problem of the location of these facilities within a distributed operating system. In order to achieve high performance the mechanism oriented facilities, i.e., process migration and remote process creation, should be placed very low in a logical architecture of an operating system. Global scheduling, employing both dynamic load balancing and static allocation, as a policy facility could be placed at a 'high level' within an architecture. This is not easily achievable in an already built system, as was demonstrated when trying to add a process migration facility within Mach [7] and Amoeba [8]. However, this goal has been successfully achieved in the RHODOS System.

2.3 The RHODOS system

RHODOS has been developed based on the client-server model, as a set of cooperating and competing processes using message passing, which are supported by the RHODOS microkernel. This microkernel based architecture is the result of our study into operating systems from the software engineering perspective [9].

RHODOS has been developed as a new and original distributed operating system, running on a bare machine. Currently RHODOS runs on a network of Sun 3/50 workstations connected by an Ethernet network and is presently being ported to the Sun SPARC Workstation. From the beginning of its design to the current stage of its development, RHODOS aims at providing high performance computing and good support to a user. These two objectives have been achieved by the way of a logical architecture; designing and building process migration and remote process creation facilities as an integral part of the system and providing user autonomy and friendliness through attributed names and execution transparency. A user sees his/her workstation as a working environment for their processes. However, these processes run transparently on any workstation within a distributed system.

The logical architecture of RHODOS is shown in Figure 2. There are three levels of processes supported under RHODOS, User Processes, System Servers and Kernel Servers. Each process executes in user mode and is confined to an individual address space which is controlled and maintained by a RHODOS microkernel, the Nucleus.

The highest logical level of processes within RHODOS are user processes. User processes have no special privileges and obtain services and resources by calls to the Nucleus, Kernel and System Servers.

System Servers constitute the next layer of the process hierarchy. These servers implement the "Policies" of a distributed operating system. Similar to user processes, these servers have access to resources via the standard system calls. However, System Servers utilise privileged communication with Kernel Servers. For example, the Global Scheduler is the only process that can issue requests to the

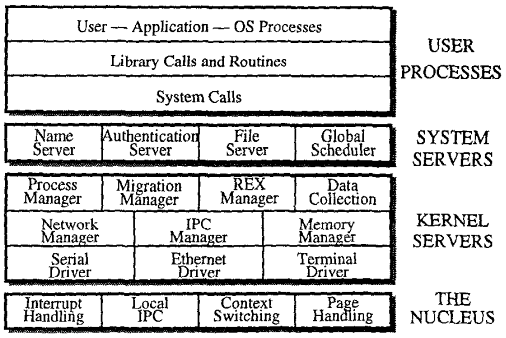

Figure 2: Logical Architecture of RHODOS

Migration Manager.

Kernel servers, which support the system servers by implementing the "Mechanisms", form the lowest layer provided within the RHODOS process hierarchy. These servers have the highest level of privileges. In addition to standard system calls the Kernel Servers have access to privileged system calls which allow the modification of data buffers and structures within the Nucleus.

2.4 Properties of RHODOS

RHODOS demonstrates the following main properties:

- Network transparency — the whole network of workstations/distributed system looks like a single powerful machine rather than as a set of connected workstations.
- User Autonomy — a user feels that he/she is dealing with their own workstation and are accessing only private resources.
- Global Scheduling — in order to balance load to improve performance and resource utilisation, process migration and remote process creation are activated by dynamic load balancing or static allocation. The Load Balancing server is responsible for making a decision *when* to move *which* process *where*. The Static Allocation server is responsible for making a decision on *where* to create a process.
- Process Migration and Remote Process Creation — a running process can be moved to a remote workstation and a process can be created on a workstation dictated by the global scheduler.

3 Process migration and remote process creation

In Section 2.3 it was shown that process migration and remote process creation facilities have been designed and implemented as two Kernel Servers, called the Migration Manager and Remote Execution (REX) Manager, respectively. This section presents these managers and their cooperation with the other servers of RHODOS.

To improve the performance of parallel execution, whilst ensuring efficient resource utilisation and maintaining a simple programming model, a number of specialised support mechanisms have been provided within RHODOS.

The first support mechanism provided is an extremely rich and flexible memory referencing system, that supports a variety of memory management and sharing facilities. These mechanisms include copy-on-write and copy-on-reference, which support efficient copying between local and remote shared memory regions. Shared memory occurs when a process is created or duplicated over a number of remote nodes. These support services are provided through the RHODOS space entity and are detailed in [10] and [11]. Through the utilisation of these specialised memory support services, a performance gain can be achieved from the duplication and creation of processes. This gain is from removing the need to completely copy the address space of a duplicated process from a source workstation to the destination. Thus, access to the physical memory that forms the address space can be achieved through copy-on-reference and copy-on-write mechanisms which are more efficient than direct copying.

The second support mechanism relates to concurrently executing processes and is the provision for group communication between processes [12]. A group communication mechanism improves performance by replacing direct communication with 1 to N communication paths and also makes programming easier.

3.1 Process migration

3.1.1 Transaction-based process migration: Each process in RHODOS has at least two ports for communication and at least four spaces which map the text, data and stacks to physical memory [9]. A space is a continuous region of memory which has a known start address and a known length. Spaces have a number of attributes that allow them to be named, protected and shared. Files in RHODOS can be memory-mapped. If a file is memory-mapped then it is linked (from disk to memory) via a space. This system allows one single mechanism to handle migration of memory mapped files and the text, data and stack spaces. To migrate a process in RHODOS, involves migrating the: process state, address space, communication state and any other associated resources with the co-operation of the appropriate server.

To perform reliable process migration in RHODOS, it is necessary to do the following transaction based sequence of operations [13]:

- The source workstation sends a message to transfer the process identification and which resources are about to be migrated. This message is considered the start of the migration *transaction;*
- The source requests the Process Manager to ensure the process is in a fit state to migrate (on the appropriate queue, not in a system call, etc.). Then place the process on the frozen queue, then transfer process state to destination;
- The source requests the appropriate servers to transfer process resource details, e.g., File Server, Space Manager, IPCM, Device Manager. Note that when the IPCM receives this request, it will freeze the process' ports;
- The destination Migration Manager receives the control information and creates state reflecting that a migration is underway. As each resource is transferred, the appropriate server notifies the Migration Manager. Once all

resources have been received, the Migration Manager sends the result to the source Migration Manager;

- The source Migration Manager waits until the destination Migration Manager sends a reply with the result, then removes the redundant process. The destination then starts the migrated process. This message is considered to commit the *transaction*.

In RHODOS, the Space Manager transfers the address space. The address space can be transferred with one of many transfer strategies (selectable for each migration without recompilation) [13]. The InterProcess Communication Manager (IPCM) [12] in RHODOS manages a process' ports and all remote messages. When a process' ports are suspended, all messages (local and remote) are forwarded onto the IPCM which queues them until they can be forwarded to the process.

3.1.2 Execution of process migration: Figure 3 shows how process migration in RHODOS is executed. Firstly once the Migration Manager has been notified of *which* process to migrate *where* (at time t_0) the Migration Manager contacts the Process Manager to request that the process' state can be transferred to the destination computer. Once the Process Manager ascertains that the process is a valid migration candidate, it freezes the process and then encapsulates the process' state into a message and sends the message to the destination Process Manager. Then the Process Manager sends an acknowledgment to the Migration Manager to inform it that the state of the process has been sent.

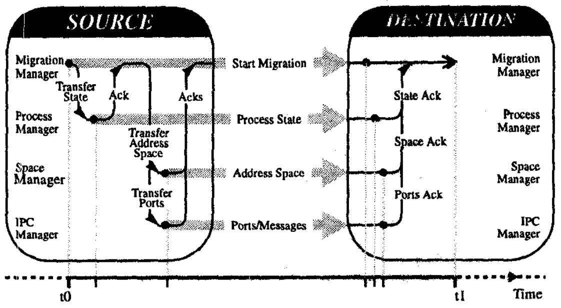

Figure 3: First Phase of Process Migration

The Migration Manager then sends out requests to both the Space Manager and the IPC Manager to transfer the memory and communication details of the process, respectively. Once the Space Manager receives the request, it encapsulates the memory of the process into a message and sends this to the destination Space Manager. Once this has been done, the Space Manager sends an acknowledgment to the Migration Manager to indicate that the process' memory has been sent. Concurrently the IPC Manager will receive the request to transfer the communication details of the process. The IPC Manager freezes the ports, encapsulates the ports and any messages on them and transfers these to the destination IPC Manager, then an acknowledgment is sent from to the Migration Manager to indicate that the process' communication details have been migrated. Once all these acknowledgments have been accepted (at time t_1) the source Migration Manager knows that the process has been migrated to the destination. Note that as the Space Manager and IPCM are contacted concurrently and due to the size of the process' address space and communication details, the acknowledgments they send to the Migration Manager can be received in any order.

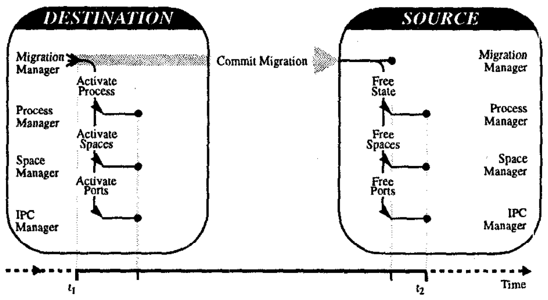

Figure 4: Second Phase of Process Migration

At time t_1 there exists a copy of the migrant process on both source and destination. Thus, the Migration Managers on both computers must confer and commit the migration (or cancel it if problems have occurred). The destination Migration Manager (at time t_1) sends an acknowledgment to the source Migration Manager, see Figure 4. Once this message has been sent, then the migration can be committed on both source and destination. For the destination this means sending a message to the local Process, Space and IPC Managers to activate the process' resources. Once these messages are received and processed, the process will start executing on the destination node. Once the source Migration Manager receives the commit migration message, then it sends three local messages to the Process, Space and IPC Managers to free up all the migrated process' resources. Upon the reception and processing of these messages the migrated process' resources will be removed and the process will no longer exist on the source computer.

3.2 Remote process creation

3.2.1 Cooperation of REX with kernel servers: The REX Manager has two interfaces, the first to the Global Scheduler and the second to a user process. The interaction of the REX Manager with the respective user process, Global Scheduler and Kernel Servers is shown in Figure 5 and Figure 6. Interaction with the file server is only initiated when a request to create a new instance of a process is received.

The REX Manager is required to coordinate the resources on behalf of the calling user process. In the coordination of user process request, the REX Manager must ensure that errors or fault conditions encountered are correctly handled. This may involve the retrying of requests when it is a temporary resource shortage problem or the flow back of error messages to the user process when more serious faults are encountered.

Here we are interested in the REX Managers interface with the Global Scheduling Server. In this relationship, the REX Manager acts as a mechanism for the static allocation component of the Global Scheduler. The Global

Scheduler informs the REX Manager that requests for process creation should be made locally or made on a remote workstation. A destination or a list of destination workstations is provided to the REX Manager and as when a request for a `process_create()` arrives, it is forwarded to the REX Manager on the respective destination workstation.

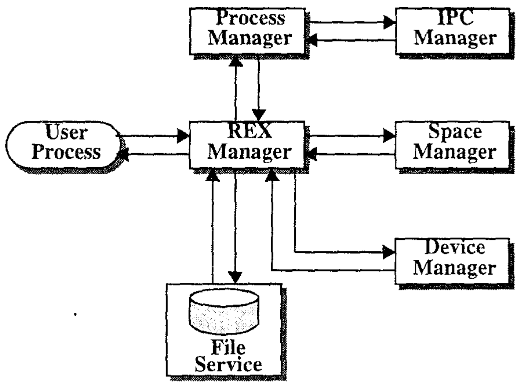

Figure 5: Local Process Creation

In this situation, the REX Manager on the source workstation acts as a peer entity in the creation of the process. As with the single workstation case, the destination workstation ensures the respective kernel servers are coordinated properly in the creation of the new process.

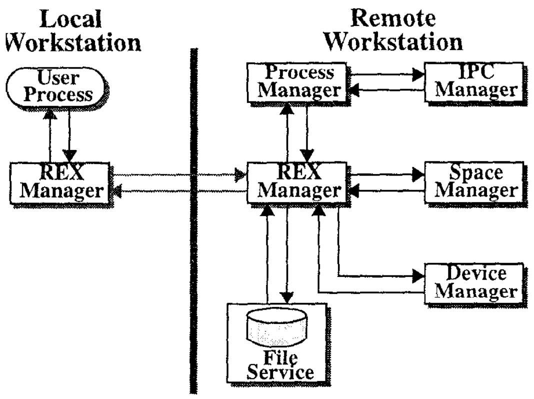

Figure 6: Remote Process Creation

The REX Manager has an interface to the file system such that creation of processes can be achieved. In the RHODOS implementation, the full distributed file system is still under development and testing. An intermediate file service was implemented on the UNIX platform and communicated directly with the RHODOS environment. Known as RUFS [14], it provides the simple interface to the file system, such that an executable image can be opened and read from.

3.2.2 Transaction-based remote process creation: Figure 7 highlights how processes creation is performed in RHODOS. The process creation model transparently performs both local and remote process creation and execution. The user process requests *which* process requires creation and notifies the REX Manager which has interacted with the Global Scheduler to receive information on the location (or locations) of *where* the process creation is to take place, at time t_0. Upon receiving the request, the REX Manager commences the process creation *transaction*. If a local process creation is to be performed, the local REX Manager will commence the allocation of resources from the local workstation. If a remote process creation and execution is required, the local REX Manager contacts the peer REX Manager on the remote destination. The remote peer REX Manager will now perform the process creation on behalf of the origin REX Manager by allocating resources on the destination host.

Once the destination has been ascertained and contacted the selected REX Manager on the destination workstation requests the local Process Manager to allocate the data structures (i.e., process control block, Process SName, etc.) for a new process. The Process Manager replies with an SName which is used to reference the newly created process. Concurrently, the REX Manager requests the Remote File/Cache Server for the retrieval of the process' image. The process image includes the text and data region of the process. In the next stage the REX Manager requests the local Space Manager for the creation of the process' spaces (i.e., data, text, user and master stacks) for the new process. The created spaces are in the newly created process' address space and not within the REX Manager's address space. Therefore a request is issued to the Space Manager for the newly created text and data spaces to be attach to the REX Manager's address space such that they can be populated from the executable image on disk. Next, the REX Manager requests a space detach from the Space Manager to allow the new process to have an independent copy of its own newly populated text and data spaces. This allows the REX Manager to send a request to the Process Manager to place the newly created process onto the ready queue to allow the process to begin execution.

Finally, the REX Manager will need to acknowledge the user process requesting process creation and execution. If this process creation and execution was performed remotely the remote peer REX Manager will now notify the origin REX Manager. At this point both local and remote process creations are at the same stage. Therefore, the transaction of process creation can be committed by the REX Manger. The user process will be acknowledged with a successful process creation at time t_4.

4 Performance of Process Migration and Remote Process Creation

To be able to make a valid comparison of the performance between the two mechanisms, it is important that the issue of "what is being measured?" is made clear. Under both cases the details measured include the size of the process and the time the mechanism takes to perform its task. Each of these parameters require clarification.

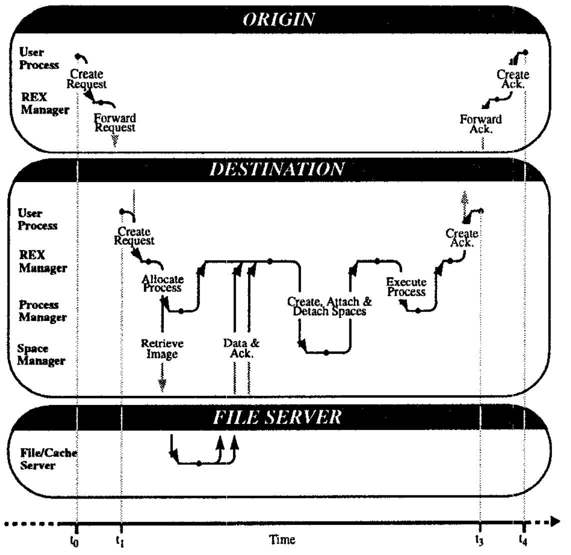

Figure 7: Process Creation

4.1 Measurement details

A measurement area that must be clarified includes the physical properties of the process being migrated compared with the process being created. In RHODOS a process is broadly divided into two resource sections: the address space resources and the process resources. The address space resources consist of the text, data and stack of a process; these spaces map directly to physical memory. The process specific resources relate to the operating system information and execution information required to support that process. Migration acts upon a process that is currently running, thus the process will have associated with it a populated address space, including variable sized user and system stacks as well as populated process resources. Therefore the transfer of the address space and process resource information from the source to the destination is required. In the remote process creation case, both the address space and process resources need to be allocated and then initialised. Thus, the text and data regions of the address space are populated from a secondary store and the stacks are simply initialised.

In our comparison, processes of equivalent sizes were used (taking only the text and data regions of the processes into account). These sizes ranged from 16 Kbytes through to over 100 Kbytes in size. In the process migration experiments it was ensured that all pages of the process were dirty for the migration; therefore the address space size is comparable to the process creation examples.

Since migration of a process is completely preemptive the size of a process' stacks at the time of migration may change depending on the level of nested procedure calls and local variables used. Also, a process in execution contains both uninitialised memory and dynamically allocated memory that is not found in an executable image on disk. Therefore the size of the process involved in process migration is always greater than the process being created and executed, which is highlighted in Figure 8. The image

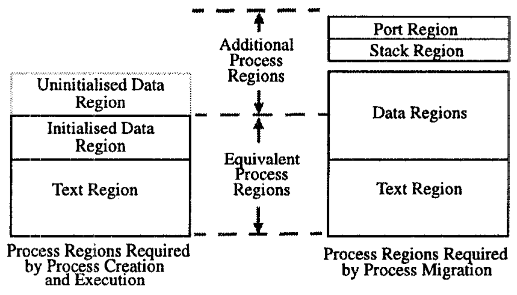
Port Region
Stack Region
Additional Process Regions
Uninitialised Data Region
Initialised Data Region
Data Regions
Equivalent Process Regions
Text Region
Text Region
Process Regions Required by Process Creation and Execution
Process Regions Required by Process Migration

Figure 8: Required Regions for Process Creation and Process Migration

of the executable process located on disk is compiled to be page aligned and therefore it is in multiples of hardware page sizes. Each process has the standard two communication ports, with no messages upon them.

The Process Migration and REX Managers both complete with the execution of a process (either a migrated or newly created process). In our experiments the time taken for each mechanism to perform its duties is taken from the initial request through to the time the (new) process begins executing.

In the process migration instance, the decision to migrate is made by the Global Scheduler and is thus transparent to the user process which is to be migrated. Therefore the starting time is when the Migration Manager receives the initial request from the Global Scheduler to perform a process migration (t_0 in Figure 3). The end time is taken as once the migrated process, on the destination machine, has been transferred to the ready queue and is able to be scheduled for execution (shown as t_2 in Figure 4).

In the process creation case [15], the request for the creation of a new process is performed by the user process, which is completely separate to the Global Scheduler. The starting time in this instance is when the user process is about to issue the process creation request (t_0 in Figure 7). The end time is once the user process has received an acknowledgment back that the new process has been created correctly (t_2 in Figure 7).

4.2 Measurement results

As was noted in [15] and [16], when processes are remotely created or migrated, one can take advantage of the fact that most creations/migrations are to workstations on the same local area network (LAN). Both RHODOS' process migration facility and remote process creation facility are capable of migrating within the one LAN with a reduced protocol stack (within one domain), or to workstations on another separate network by using a complete protocol stack (interdomain).

As was noted in Section 3.1.1, process migration can transfer the memory of a process via several different strategies [13]. In this paper, the migration mechanism wither copied the dirty pages directly to the destination or used copy on reference.

The results of process creation and both forms of process migration within the one domain are shown in Figure

9. The results of interdomain process creation and interdomain process migration are shown in Figure 10. In the key

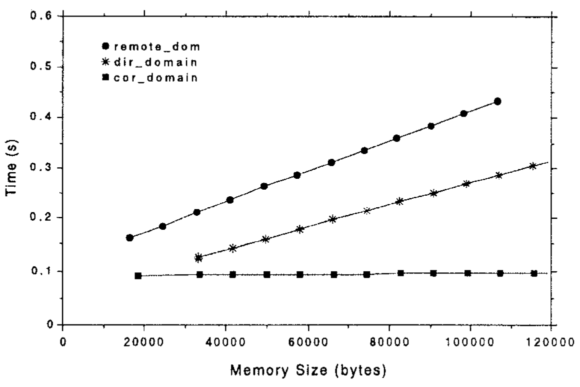

Figure 9: Domain Process Migration and Remote Process Creation

of Figure 9 remote_dom = remote process creation within the one domain, dir_domain = direct copy of the dirty pages to the destination within the one domain, cor_domain = copy on reference within the one domain. The key in Figure 10 indicates the same mechanisms utilised however across two different domains (or interdomain).

These results show that process migration is faster to perform than process creation due to the sequential behaviour of the process creation mechanisms, which is not present in the migration mechanism. Indeed, the process migration mechanism is more mature than the remote process creation mechanism and optimisations utilised by the migration mechanism can be employed by the remote process creation mechanism. These optimisations should remove up to 30 ms from the time to create a process. However, remote process creation will always be slower than process migration due to the extra messages required to communicate from the source to destination Rex Managers (see Figure 7).

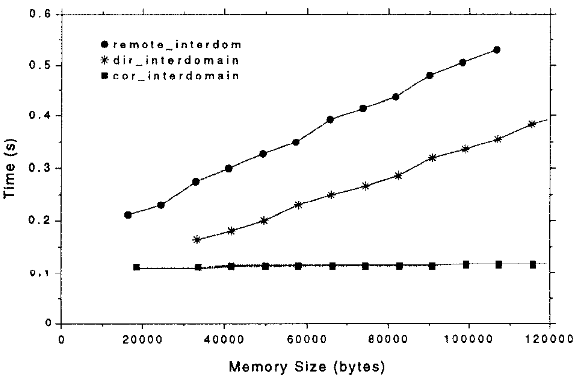

Figure 10: Interdomain Process Migration and Remote Process Creation

All sample values used in the testing follow a normal distribution. 99.5% of all sample values are found to be within 5% of the measured values.

5 Conclusion and future work

In this paper we proposed the way global scheduling should be used to improve or develop high performance computer systems based on a network of workstations/distributed systems. We have shown an architecture of a global scheduling facility employing both dynamic load balancing and static allocation. We developed the mechanisms as part of the RHODOS distributed operating system. These mechanisms have been built as kernel servers in order to achieve high performance, transparency and reliability. Because these mechanisms have been designed and implemented from the beginning of the whole RHODOS project, they have been developed quickly and efficiently, in contrast to the work reported in [7] and [8].

In order to demonstrate that the claims made by some researchers that process migration is too heavy to be used to support dynamic load balancing are unsubstantiated we carried out performance study of both process migration and remote process creation.

The performance study has shown that performance of process migration is as good as that of remote process creation and in fact it is even better. We have also demonstrated that the claims that process migration is too heavy to be used in high performance systems is fully unsubstantiated. In general, we demonstrated that both remote process creation and process migration are efficient mechanism to be used in building the global scheduling facility.

Currently, we are working on improvement of performance of the two mechanisms and initiating the project on building a global scheduling server. In the next stage we will concentrate our effort on building a system to support parallel execution on a network of workstations/distributed system using the developed mechanisms.

6 References

[1] A. Goscinski. *"Distributed Operating Systems. The Logical Design"*. Addison-Wesley, 1991.

[2] U. Trottenberg. *"Are Workstations Replacing Supercomputers or Massively Parallel Systems? Questions, Facts and a Result"*, GDM D Spiegel 1'93 D. The Journal of the German National Research Center for Computer Science (GMD).

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam. *"PVM - Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing"*. MIT Press, Cambridge, MA, 1994.

[4] D. Eager, E. Lazowska, J. Zahorjan. *"The Limited Performance Benefits of Migrating Active Processes for Load Sharing"*. Proc. of the 1988 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems, May 1988.

[5] G. Gerrity, A. Goscinski, J. Indulska, W. Toomey and W. Zhu. *"Can We Study Design Issues of Distributed Operating Systems in a Generalized Way? — RHODOS"*, Proceedings of the Symposium on Experiences with Distributed Multiprocessor Systems (SEDMS II), Atlanta, Georgia. March 1991.

[6] A. Barak, A. Braverman and O. Laden. *"Performance of PVM with the MOSIX Preemptive Process Migration"*. Institute of Computer Science, The Hebrew University, Technical Report 95-08, March 1995.

[7] D. S. Milojicic. *"Load Distribution. Implementation fro the Mach Microkernel"*. Vieweg Advanced Studies in Computer Science. 1994.

[8] W. Zhu. *"Dynamic Load Balancing on Amoeba"*. Proceedings of IEEE First International Conference on Algorithms and Architectures for Parallel Processing. Brisbane, Australia. April 1995.

[9] D. De Paoli, A. Goscinski, M. Hobbs and G. Wickham. *"The RHODOS Microkernel, Kernel Servers and their Cooperation"*. IEEE First International Conference on Algorithms and Architectures for Parallel Processing (ICA^{3}PP). Brisbane Australia. April 1995.

[10] M. Hobbs, D. De Paoli, A. Goscinski, G. Wickham and R. Panadiwal. *"Generic Memory Objects for Supporting Distributed Systems"*. International Conference on Automation (ICAUTO-95). Indore, India, December 12-14, 1995.

[11] D. De Paoli and A. Goscinski. *"Copy On Reference Process Migration in RHODOS"*. (Submitted to) The IEEE Second International Conference on Algorithms and Architectures for Parallel Processing (ICA^{3}PP-96). Singapore Australia., June 1996.

[12] P. Joyce, D. De Paoli, A. Goscinski and M. Hobbs. *"Implementation and Performance of the Interprocess Communications Facility in RHODOS"*. Proceedings of the International Conference on Networks, Singapore, October 1995.

[13] D. De Paoli. *"The Multiple Strategy Process Migration for RHODOS: The Logical Design"*. Technical Report TR C93/37, Deakin University, Australia, December 1993.

[14] M. Hobbs. *"A Simple Open File Facility for RHODOS"*. School of Computing and Mathematics, Deakin University. Technical Report C95/05, January 1995.

[15] M. Hobbs and A. Goscinski. *"A Remote Process Creation and Execution Facility Supporting Parallel Execution on Distributed Systems"*. (Submitted to) The IEEE Second International Conference on Algorithms and Architectures for Parallel Processing (ICA^{3}PP-96). Singapore Australia., June 1996.

[16] D. De Paoli and A. Goscinski. The Influence of Domain and Interdomain Process Migration on the Performance of Parallel Execution on Distributed Systems. *Proceedings of the International Conference on Parallel and Real Time Systems*, Perth Australia, September 1995.