A Concurrent Process Creation Service to Support SPMD Based Parallel Processing on COWs

M. Hobbs and A. Goscinski

School of Computing and Mathematics
Deakin University, Geelong
Victoria 3217, Australia
{mick, ang}@deakin.edu.au

Phone: +61 3 5227 2088
Fax: +61 3 5227 2454

Summary[1]

Current software environments used to support parallel processing on Cluster of Workstations (COW) are not satisfactory, do not provide complete transparency and are not specifically designed for parallel processing. In particular, the establishment of a parallel processing environment and the initialisation of parallel processes suffer from poor performance. Each parallel process of an application is created sequentially and in many cases the logon operation must be completed before remote resources could be acquired. These operations are also performed manually. We present in this paper an original approach that addresses the problem of parallel process creation. The remote workstations are acquired completely transparently and dynamically, and parallel processes are created concurrently. To demonstrate the feasibility of this approach we show a system based on RHODOS (a client/server and microkernel based distributed operating system), specifically designed to improve the performance of process instantiation and therefore able to improve the overall execution performance of parallel programs, in particular parallel process creation.

***Keywords**: Parallel Processing on COWs, Operating Systems, Concurrent Process Creation, Performance Evaluation.*

1.This work was partly supported by the ARC Grant 0504003157.

1 Introduction

Current attempts to harness the computational resources of Clusters of Workstations (COWs) for parallel processing have generally followed the approach of building an environment on top of an existing operating system such as Linux (Beowulf [1]), Solaris (NOW [2]), SunOS (Paralex [3]). These systems run PVM [4] or MPI [5] in order to exploit the collection of workstations, and provide middleware packages such as GLUinx [6], LSF [7], Condor [8] or LoadLeveler [9] to locate processes. This approach of using off-the-shelf operating systems is attractive from the initial investment point of view. Unfortunately, such solutions lack transparency which increases the programming burden on the user, require the parallel processes of an application to be created sequentially, and in many cases need the logon operation to be manually completed before remote resources could be acquired. The overheads of such solutions are also increased due to the duplication of services within the environment and the operating system as it was demonstrated in [10]. The reasons is that parallelism is managed poorly.

In order to manage parallelism to achieve high performance and to make COW based parallel systems easy to use and parallel applications easy to program, a distributed operating system should be employed. Such an approach has been advocated and demonstrated that it is feasible in both [11][18], where MOSIX based load balancing and process migration services were used to allocate processes to workstations in order to support PVM applications; and [12], where RHODOS based load balancing and remote process creation services were used to allocate processes to workstations and load balancing and process migration to dynamically balance load. However, in both cases each parallel process was created individually that generated huge costs.

A variety of parallel processing models exist that are applicable to the execution on COWs. Owing to the relatively high overheads of the networks used to interconnect workstation (when compared to bus based systems), COWs are more suited to coarse grained parallelism such as program based parallelism. Two processing models exist at this level: the data parallelism model, also called the Single Program Multiple Data

(SPMD) model; and the functional model, also called the Multiple Program, Multiple Data (MPMD) model [13]. The focus of this work is on the SPMD model of parallelism as it is simpler and more commonly employed.

The goal of this paper is to present a new system specifically designed to support parallel processing, in particular SPMD parallel processing and which is based on a distributed operating system. In particular, the concurrent process creation services provided within this system are to be presented, their operation detailed and the performance study results shown.

To achieve this goal, this paper is organised as follows. Section 2 presents the important issues relating to the execution of SPMD parallel applications on COWs and related systems. Furthermore, this section shows the logical design of a process creation service specifically designed to support SPMD based parallel processing. Chapter 3 introduces the RHODOS distributed operating system and details the identification of suitable, idle or lightly loaded, workstations. Furthermore, the implementation issues encountered when developing the concurrent process creation service are elaborated. Chapter 4 presents the experimental environment that was used to investigate the performance of the RHODOS creation service the performance results obtained, and a summary of the results. A conclusion that highlights the achievements of this research and their assessment are presented in Chapter 5.

2 The Design of a Concurrent Process Creation Service

The goal of this section is to identify the problems experienced when executing SPMD based applications on COWs and to provide the background for the logical design of a concurrent process creation service.

2.1 SPMD Based Parallel Processing on COWs

As presented in the introduction, the course grained SPMD model of parallelism is ideal for execution on COWs due to the high ratio of computation to communication. An SPMD parallel program can be divided into three distinct phases, the initialisation, execution and termination phases, as shown in Figure 1. The initialisation phase involves the instantiation of a set of identical child processes and the distribution of the data (work) to each of these child processes. The next phase involves each of the child processes executing in parallel on their allocated data. It is during the execution phase that the actual work on the problem data is performed, and increasing the number of child processes executing in parallel on the problem data can improve the overall execution time. Finally, in the termination phase each child completes the work on their data and then returns the result back to the parent process before exiting. The focus of this work relates to the initialisation phase of the parallel program, more specifically, the process creation method of instantiating the set of child processes.

The overall execution time of a parallel program is limited by the amount of sequential processing it contains. The sequential processing within an SPMD parallel program is commonly dominated by the initialisation phase and more specifically the instantiation (creation) of the child processes.

[Figure 1: Structure of an SPMD Based Program]

Process creation is the invocation and execution of a child process from an executable image located on disk. A process is an entity that has a number of physical and logical resources associated with it. The physical resources relate primarily to the memory usage, communication end points and buffers. The logical resources relate primarily to the execution state of the process. The process creation service is an operation that requires substantial computation in order to marshal the various memory, process and communication resources that combine to form a child process. This service is even more time consuming if a process is created on a remote workstation. The situation worsens when n child processes are sequentially created on n workstations of a virtual machine. It is clear that if the overhead of process creation is reduced the program enters the parallel execution phase more quickly and thus enabling the overall execution time to be improved with higher levels of parallelism.

2.2 Related Systems

A number of currently available systems provide support for the execution of SPMD based parallel programs on COWs. The most common approach taken with these systems is to provide a support environment that executes on top of an existing network operating system. These environments take the form of specialised libraries and server processes, the most well known of these systems is Parallel Virtual Machine (PVM) [4]. The PVM system is the result of a long running project to provide an environment that uses a collection of (possibly heterogeneous) workstations and presents the programmer with a set of consistent communication, execution and synchronisation primitives.

An improvement to the common approach of the PVM system executing on top of an existing operating system is through the extension and enhancement of the underlying operating system to provide services and features supporting parallel processing. This approach has been taken by the Beowulf [1], Paralex [3], NOW [2] and MOSIX [11] systems.

The Beowulf system exploits distributed process space (BPROC) [14] in order to manage parallel processes. There are two basic issues of BPROC supporting distributed processes, ghost processes that represent remote processes on remote computers, and the method of starting processes. Processes can be started on remote computers if the logon operation into that remote computer was completed successfully. Starting processes in Beowulf is only done via *rsh*, sequentially, although these two features are hidden from the user by PVM and MPI. Another weakness of Beowulf is that BPROC does not address resource allocation or load balancing. The project does not attempt, at this stage, to provide transparent process migration.

Paralex [3] is a complete parallel application programming environment with run-time support features and a simple to use graphical interface allowing programmers to define, edit, execute and debug parallel applications. Paralex automatically generates code for distributing computation; and process replication. The Paralex system employs passive replication to support fault tolerance within a parallel application and as a method to support run-time based dynamic load balancing. Paralex is built as a set of daemon and controlling processes, that execute on existing operating systems. Since Paralex has been developed at the user level, no modifications are required to the operating system.

The NOW system is based on the Solaris operating system from Sun Microsystems and combines specialised libraries and server processes with enhancements to the kernel itself in the form of the scheduling and communication kernel modules. The enhancements to the operating system have been in the form of a global operating system layer (GLUnix) to provide network wide process, file and virtual memory management [6]. The services of the current GLUnix version have been built at the user level through the support of a master process, daemon processes and GLUnix libraries. The limitations of running the system in user mode were identified in [6], and relate to the loss of transparency introduced when executing processes remotely. To remove this problem, it is intended that future versions of GLUnix take advantage of the dynamically loaded kernel modules associated with process management, memory management, scheduling and I/O services, such that processes are provided with a complete, transparent and global view of the network of workstations.

The current version of the MOSIX system is a result of the extension and modification of the Linux operating system to produce a fully distributed operating system. The MOSIX system provides enhanced and transparent communication and scheduling services within the kernel, and also employs PVM to provide the high level parallelism support. MOSIX is based on the Unique Home-Node (UHN) model [15], where all of users processes appear to be run on the local workstation, but due to load imbalances, process migration may transparently migrate processes to a remote workstation. Once migrated, the process is divided into two parts, the deputy component which remains on the local workstations or UHN, and a remote component which encapsulates the running process on the remote workstation. The deputy component provides a method of handling all UHN dependent functions such as communication, signals and site dependent system calls.

A common and serious problem exists with the PVM, Beowulf, NOW and MOSIX systems, which is amplified when they attempt to support parallel processing. Each of these systems only supports the creation of single processes. Although the primitives provided by each of these systems enable the user to request multiple processes to be created, internally they are only created one at a time. This problem is the result of the reliance each of these systems have on the underlying network operating systems, which were designed and implemented to only support the creation of single processes. Therefore, if a given SPMD parallel program requires 100 child processes to be created, then the single creation service needs to be called 100 times. Another major problem of these systems is that they do not provide complete transparency with the exception of MOSIX, and a virtual machine is not set up automatically and dynamically.

2.3 Process Creation Service

To design a process creation service specifically for SPMD based parallel processing, a number of requirements need to be addressed, including:

- Multiple Creation of Processes — it must be possible to create concurrently many instances of a process on a workstation or over many workstations;
- Scalability — the proposed service, to take full advantage of available parallelism, must be scalable to many workstations; and
- Complete Transparency — the proposed service must hide from the user all location and management elements, such that the user (programmer) is unaware of the location of the parent and child processes.

The SPMD model of parallel processing has two main characteristics which simplify the provision of these requirements. The first and most important characteristic is that a SPMD parallel program is composed of multiple identical copies of the one child process. Each of these child processes begins execution at the same point within the program. Secondly, each of the child processes within a SPMD parallel program do not interact with each other, simplifying the provision of the transparency requirement.

With the traditional single process creation service, the executable image of the child process is required to be downloaded from disk for every instance of the child. The fact that each child is identical and that many child processes may be instantiated on a given workstation, enables memory sharing between the child processes to be employed and thus reducing the creation time and improving memory usage. The text regions of the child processes can be shared “read only”, while the data regions can be shared “copy on write”.

[Figure 2: SPMD Based Parallel Program on a COW]

It may also be necessary to create multiple instances of the child process on many workstations remote to the workstation on which the parent is executing. Therefore the creation service provided must not only be capable of creating many processes on a single workstation, but must also be able to create them on many workstations. This situation is illustrated in Figure 2, where *n* child processes are created on *n* workstations. The placement of the child processes on the various workstations within the COW can greatly influence the performance of the overall program due to load imbalances [16]. If child processes are mapped to workstations that are heavily loaded whilst other workstations are lightly loaded or idle the overall performance obtainable from the program will be diminished. It is desirable for the child processes to mapped to a set of workstations such that the overall load of the COW remains balanced. When distributed evenly across the COW each child process is provided with an equal share of the available computational resources and thus capable of achieving higher performance. The mapping of processes to workstations is therefore closely linked to the process creation service since it is at this time that the new child processes are brought into existence.

2.4 Logical Design of a Process Creation Service

A process is composed of memory and process related resources. For this reason each of these resources could be managed by separate servers [17]. Furthermore, a process creation server could also be provided to marshal the allocation of each of the resources to form a new child process. We propose here that to improve the flexibility, modularity and implementability of a process creation service the client/server model should be employed, where different services (management of resources) are provided by different server processes.

Figure 3 illustrates the interaction between the servers involved in the creation of a child process, including the Memory Server, Process Server and File Server. Also presented in this figure is the Process Mapping Server which is required to map processes to workstations at their creation, thus assisting to maintain a COW wide balanced load.

The process creation service presented shows the traditional single process creation. It was stated earlier that to efficiently support SPMD parallel processing on COWs more advanced services are required. The single process creation service can be extended using memory sharing operations to allow many child processes to be created on the same workstation with the executable image only being downloaded once. The multiple process creation service involves the allocation of the required number of child process entries and the first child process being created following the single process creation method. After completing this operation, each successive child process is produced as a copy of the memory of the first child. The text regions are shared “read only”, while the data regions are shared “copy on write”. This enhancement offers the benefits of reduced creation time due to the executable image only being downloaded once and improves memory utilisation due to the sharing of physical memory.

[Figure 3: Server Interaction to Create a Single Process]

When many processes are required to be created on many workstations, the multiple process creation operation can then be performed on each workstation selected by the Process Mapping Server. Using this method, the executable image is required to be downloaded for each workstation involved in the multiple process creation. As with the single process creation, it is possible to enhance the multiple process creation method to reduce the overall creation time and therefore improve the overall execution performance. To enhance the multiple process creation service when many workstations are involved requires the employment of group communication. Instead of the executable image being sent from the File Server to each participating Process Creation Server, group communication can be used and with a single message the image can be sent to all servers, resulting in group process creation.

3 The Implementation of an Advanced Process Creation Service

This section demonstrates the feasibility of the logical design of the advanced process creation service proposed to improve the overall execution of SPMD applications on COWs.

3.1 RHODOS Parallelism Management System

The design of the process creation service presented in Section 2 has been implemented within the RHODOS distributed operating system [17]. Unlike the current approaches of building environments on top of existing networked operating systems or the partial extension to an existing network operating system, the approach taken here was to build the process creation service as inherent part of a purpose built distributed operating system.

[Figure 4: RHODOS Parallelism Management System]

The RHODOS process creation service is a component of the RHODOS Parallelism Management System [13], [16], the structure of which is presented in Figure 4. As identified in Section 2.3, the process mapping and process creation servers are embodied in the RHODOS Global Scheduler and RHODOS Execution Manager (REX), respectively. These two server processes form the core of the RHODOS Parallelism Management system but rely on the services of the Inter Process Communication (IPC), Process, Space (Memory) and File Managers to perform the required operations.

The role of the Global Scheduler is to ensure the load of the entire COW remains balanced, and in particular, it provides a process allocation role when mapping processes to workstations at their instantiation. To achieve this the Global Scheduler, if centralised, uses load information collected about the load conditions of each of the workstations within the COW on which it bases its mapping decisions, or if distributed, exchanges information about computational load among peer Global Schedulers running on each workstation of the COW. The current implementation of the RHODOS Parallelism Management System employs a single, centralised Global Scheduler that uses event notification to record the current loads of the workstations within the COW. The events here relate to load changes on a workstation, where a process enters an executable state (created) or leaves an executable state (exits).

The Global Scheduler cooperates with the REX Manager to collect computational load information and to enact the process creation decisions. The REX Manager exists on each workstation within the COW and provides the interface between the user (parent) process and the RHODOS Parallelism Management System. The parent process sends the REX Manager a request to create a given number of child processes. The REX Manager obtains from the Global Scheduler the location of the workstations on which these child processes should be created. The REX Manager then cooperates with it's peer REX Managers on the selected remote workstations to perform the process creation.

As indicated in Section 2.1, it is necessary to create multiple instances of a child process on a workstation and/or multiple workstations. The REX Manager is capable of creating many copies of a child process on a workstation with the support of the Process and Space Managers, and with the use of group communication is able to concurrently create child processes over many workstations using a single message. The remainder of this section is devoted to showing how processes are created in RHODOS using, single, multiple and group methods; for both local and remote creation.

3.2 Single Local and Remote Process Creation

The standard method available to create processes is through single process creation [12]. The child process, depending on the mapping decision of the Global Scheduler, can be created on the same workstation as the parent, 'Local Creation'; or on a different workstation, 'Remote Creation'. The RHODOS Parallelism Management System ensures that the location of the child process is hidden from the parent process, all interaction (communication and coordination) between the parent and child process is completely transparent.

The interaction between the parent process, REX Manager, Global Scheduler, Process Manager, Space Manager involved in the creation of a new, single, child process on the same workstation as the parent (local) is presented in Figure 6. This figure also shows the relative order of the messages required to create a process with the RHODOS Parallelism Management System. The Global Scheduler and File Server are shown on a separate (dedicated) workstation.

The interaction (and order of messages) between the parent process and the respective managers involved in the remote creation of a single process is presented in Figure 6. In this case the Global Scheduler responds to the location request of the local REX Manager with the address of a remote workstation. The REX Manager on the local workstation forwards on the request to its peer REX Manager on the selected workstation. The actions undertaken by the REX Manager on the remote workstation are identical to that of a local process creation. The only difference is that the acknowledgment of the completed creation is sent back to the REX Manager on the local workstation, which registers the new remote child process with the local Process Manager and then replies with the creation result back to the parent process.

[Figure 5: Order of Events in the Single Local Process Creation]

[Figure 6: Order of Events in the Single Remote Process Creation]

3.3 Multiple Local and Remote Process Creation

The basic single process creation service was extended in the RHODOS Parallelism Management System to support the creation of multiple child processes on one workstation by employing memory sharing. Figure 6 shows the interaction and order of messages involved in the creation of *n* multiple child processes on two workstations.

These workstations are selected by the Global Scheduler and in this case are the local workstation which is required to have *m* child processes created and a remote workstation which is required to have *n-m* child processes created (where $n > m$). The order of operations are similar to the single process creation, although after the first child has been created, the remaining child processes are formed from duplicates of the first child's memory. This involves an extra call to the Space Manager to duplicate the remaining children's memory.

[Figure 7: Order of Events in the Remote Multiple Process Creation]

It is important to note here that for each workstation selected by the Global Scheduler as a recipient of a child process, the File Server is contacted with a request to download the executable image of the child process. As the number of workstations increases, it is possible that the File Server can form a bottle neck within the process creation that limits the performance of the operation.

3.4 Group Process Creation

The limitation found with the multiple process creation service was solved with the employment of group communication between the File Server and the REX Managers involved in the process creation operation. Each REX Manager and the File Server join a communication group where a message sent to the group is received by all members of the group. In the group creation method, the local REX Manager acts as a coordinator for the overall creation and is the only REX Manager that requests the image of the child process to be downloaded. In this case, the File Server responds by sending the executable image to the REX Managers group, rather than to the local (requesting) REX Manager. Therefore, the image of the child process is received by all participating REX Managers with the sending of a single message. Although the communication in the RHODOS COW is based on a reliable message passing protocol, messages may be delayed due to lost packets. In the case of a delayed message, if the image of the child process is received by the remote workstation before the forward create message is received it is held for a short period of time waiting for the forward create message to be received. If the forward create message is received, the group creation continues normally, otherwise an error message is generated and passed back to the local workstation and the process creation cancelled.

The interaction and order of the messages involved in the group creation of a set of child processes is shown in Figure 6. The distinction between this operation and the multiple creation operation shown in Figure 6 is that only one request is issued to the File Server and only one message of the executable image is sent in a reply. Therefore, the bottleneck problem faced with the previous method is avoided here as the number of workstations increases.

[Figure 8: Order of Events in the Group Process Creation]

4 Experiments and Performance

The implemented system presented in the previous section was tested on the RHODOS COW. This cluster is composed of 13 Sun 3/50 workstations interconnected by a shared 10 Mb/s switched ethernet network. One of the workstations within the COW is dedicated as a File Server and the remaining 12 workstations are used for normal user computation. The structure of the RHODOS COW is presented in Figure 9.

We present in this section the performance results obtained from testing each of the three creation methods available within the RHODOS Parallelism Management System. To examine the performance of these services a sample SPMD based parallel program was developed and the time to instantiate a number of child processes was measured. The experiment was performed with the number of child processes being varied from 1 through to a total of 20 child processes. The number of workstations involved was also varied from 1, 2, 4, 8 and 12. Each experiment was performed 20 times and the average result presented.

[Figure 9: The RHODOS COW]

The performance results obtained from using the single process creation service is presented in Figure 10. From these results it is clear that the single process creation service provides, as expected, the worst performance. This can be attributed to the extremely high overhead of having to repeatedly call the process creation code for each child created. Therefore, a linear performance result is observed as the number of child processes created increases. A slight overall increase can be observed when the number of workstations increases which can be attributed to the overhead required to forward on the creation request from the local REX Manager to the peer REX Manager on the remote workstation.

A considerable performance improvement is observed when memory sharing between multiple child processes on the same workstation is employed (Figure 11). This is shown in the results of the multiple process creation service. Here, a dramatic levelling off is obtained when more than one process is created on a given workstation. These results highlight the performance improvement obtained from memory sharing over downloading the image form disk. As the number of workstations used in the experiment increases, the execution time also increases which can be attributed to the downloading the image to each new workstation.

The group process creation service provides the best performance results obtained from the three different services (Figure 12). The overall performance is inherently linear with only a slight step found when two or more workstations are used. This step can be attributed to the change from a local process creation to a remote process creation and therefore the need to forward on the remote process creation request to each of the remote workstations.

[Figure 10: Single Process Creation]

[Figure 11: Multiple Process Creation]

[Figure 12: Group Process Creation]

5 Conclusion

If a set of parallel processes of an application executing on a COW can be created automatically and transparently, application programmer will be relieved from time consuming, error prone and irrelevant activities. If the overhead of creating this set of parallel processes executing on a COW can be reduced, the overall performance obtainable form the parallel program can be improved. This can be achieved if parallelism management is addressed and provided.

We presented in this paper a number of prominent systems including, PVM, Beowulf, Paralex, NOW and MOSIX, which attempt to address the management of parallelism on COWs, with the aim of harnessing the vast computational resource they hold. Unfortunately, each of these systems relies on an existing network operating system to create the instances of the child process. These operating systems only support the creation of single processes, which can severely limit the performance of a parallel programs, especially as the number of child processes and workstations increase. Furthermore, some of the systems (e.g., Beowulf) require the logon operation on remote computers to be completed before parallel process creation could be performed.

We demonstrated that the RHODOS Parallelism Management System addresses these problems by employing services of a distributed operating system. The RHODOS Parallelism Management System, presented in this work, provides a number of effective services that were developed specifically with SPMD parallel processing in mind. First of all, the system identifies the available (lightly loaded or idle) computers and allows them to be used to create remote processes transparently. Furthermore, this system supports the multiple creation of processes on a given workstation and provides group process creation over a number of workstations. Process images are distributed in parallel to all workstations of the virtual machine involved with the execution of the SPMD parallel program. These operations are carried out completely transparently and automatically, thus they relieve the application programmer from irrelevant activities. It is evident from the performance results that the RHODOS group process creation service provides dramatic performance improvements over the standard single process creation service, especially as the number of workstations increases.

6 References

1. D. Ridge, D. Becker, P. Merkey and T. Stirling, ‘Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs’, *Proceedings of the IEEE Aerospace*,

1997.

2. T. Anderson, D. Culler and D. Patterson, 'A Case for Networks of Workstations: NOW', *IEEE Micro*, **15**(1), 1995.

3. R. Davoli, L-A. Giachini, O. Babaoglu, A. Amoroso and L. Alvisi, 'Parallel Computing in Networks of Workstations with Paralex', *IEEE Transactions on Parallel and Distributed Systems*, **7**(4), 371-384, 1996.

4. D. Beguelin, J. Dongarra, A. Giest, R. Manchek, S. Otto and J. Walpole, 'PVM: Experiences, Current Status and Future Directions', Tech. Rep. CSE-94-015, Oregon Graduate Institute of Science and Technology, April 1994.

5. Message Passing Interface Forum, 'MPI: A Message-Passing Interface Standard', *International Journal of Supercomputer Applications* (Special issue on MPI), **8**(3/4), 1994.

6. P. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat and T. Anderson, 'GLUnix: a Global Layer for a Network of Workstations', *Software—Practice and Experience*, **28**(9), p. 929-961, 1998.

7. S. Zhou, 'LSF: Load Sharing and Batch Queuing Software', *Platform Computing Corporation*, North York, Canada, 1996.

8. M. Litzkow, T. Tannenbaum, J. Basney and M. Livny, 'Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System', Tech., Rep. #1346, U. of Wisconsin-Madison Computer Sciences, April 1997.

9. IBM, 'IBM LoadLeveler: User's Guide', *International Business Machines Corporation*, **3**, 1993.

10. J. Rough and A. Goscinski, 'Performance Comparison of the RHODOS PVM and the UNIX PVM', (To Appear) *Proceedings of The International Symposium on Parallel Architectures, Algorithms and Networks* (ISPAN-99). CS-Press. ISPAN-99, Perth, June 1999.

11. A. Barak and O. La'adan, 'The MOSIX Multicomputer Operating System for High Performance Cluster Computing', *J. of Future Generation Computer Systems*, **13**(4-5), 361-372 (1998).

12. M. Hobbs and A. Goscinski, 'The RHODOS Remote Process Creation Facility Supporting Parallel Execution on Distributed Systems', *J. of High Performance Computing*, **3**(1), 23-30 (1996).

13. A. Goscinski, 'Towards and Operating System Managing Parallelism of Computing on Clusters of Workstations', (Submitted to) *Parallel Computing*.

14. E. Hendriks, 'BPROC: Beowulf Distributed Process Space', Web Address, *http://www.beowulf.org/software/bproc.html*, April 1999.

15. Mosix4Linux

16. M. Hobbs, 'The Management of SPMD Based Parallel Processing on Clusters of Workstations', PhD. Thesis, School of Computing and Mathematics, Deakin University. August 1998.

17. D. De Paoli, A. Goscinski, M. Hobbs and G. Wickham, 'The RHODOS Microkernel, Kernel Servers and Their Cooperation', in *Proceedings of The IEEE First International Conference on Algorithms and Architectures for Parallel Processing* (ICA3PP-95), Brisbane, Australia, April 1995.

18. A. Barak, A. Braverman, I. Gilderman and O. La'adan, 'Performance of PVM with the MOSIX Preemptive Process Migration', in *Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, June 1996, pp 38-45.

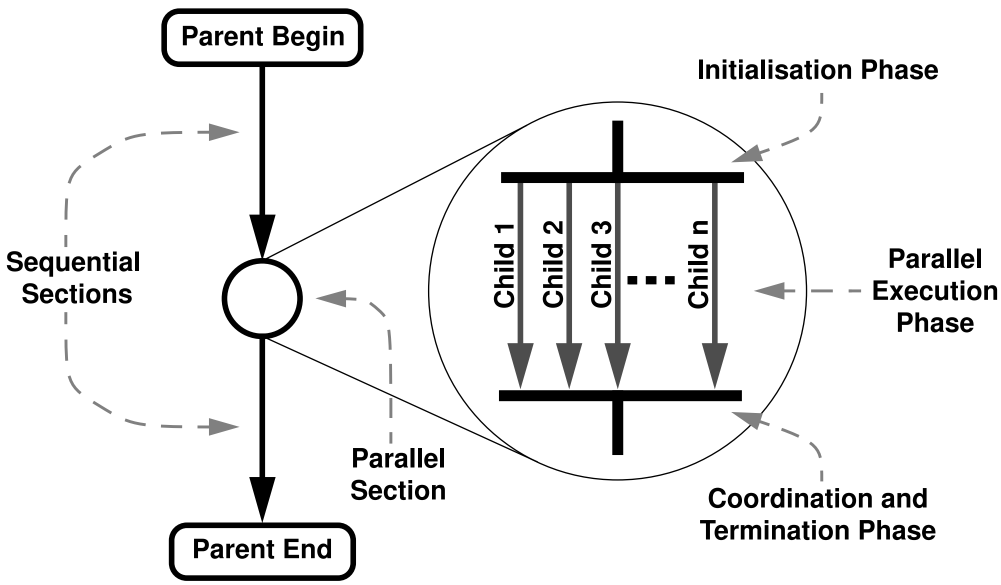

Figure 1: Structure of an SPMD Based Program

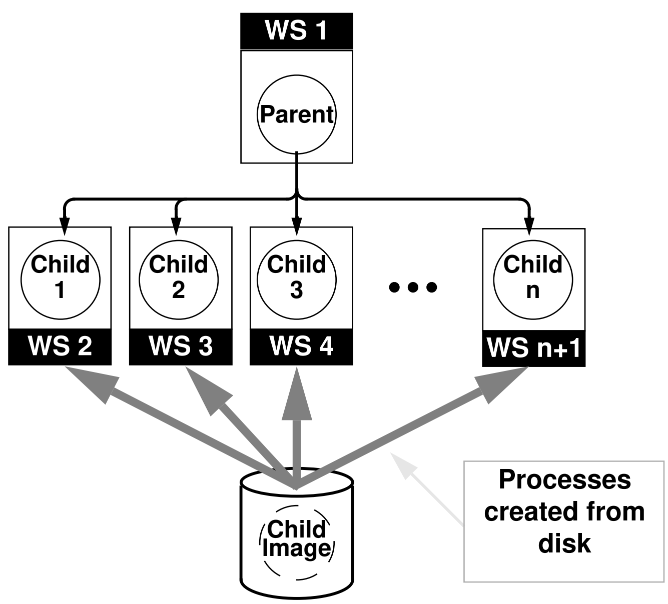

Figure 2: SPMD Based Parallel Program on a COW

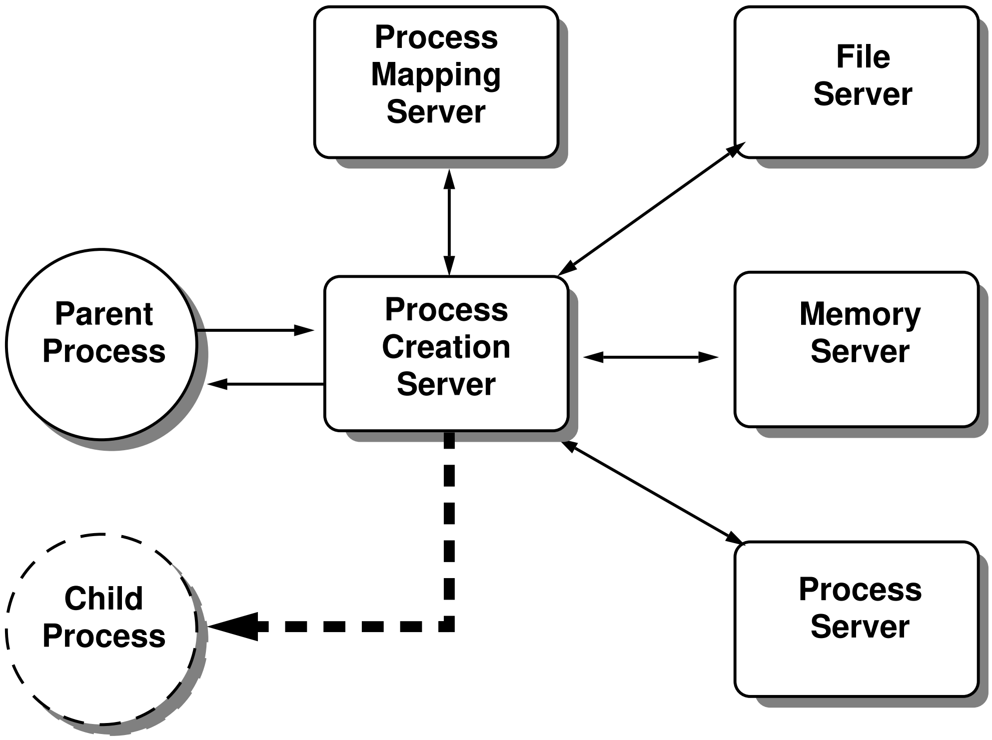

Figure 3: Server Interaction to Create a Single Process

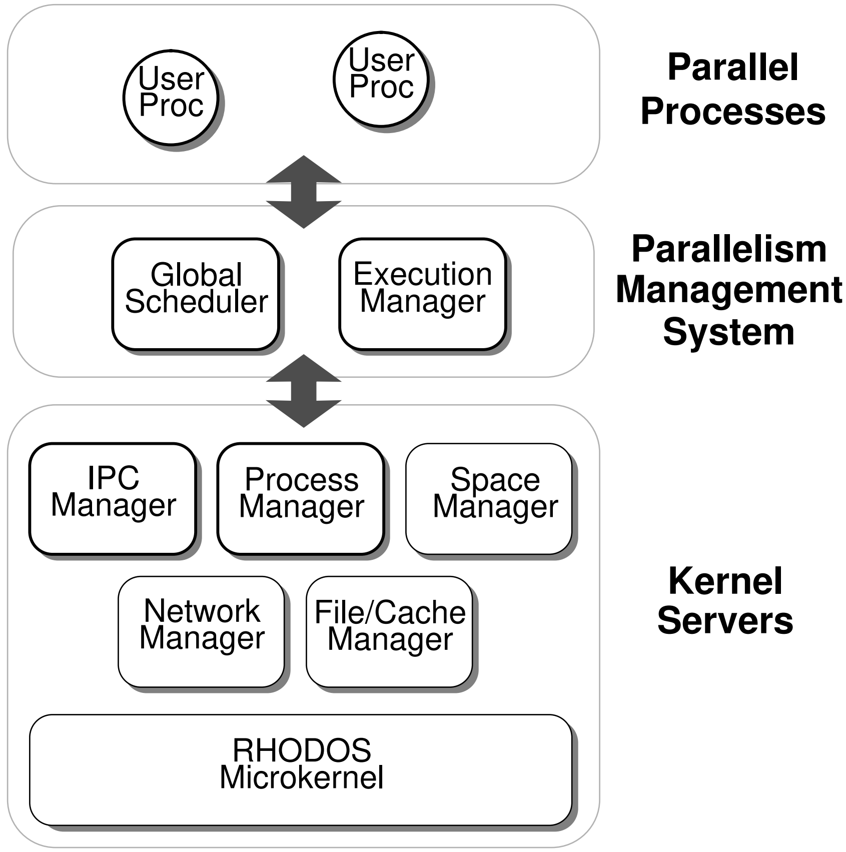

Figure 4: RHODOS Parallelism Management System

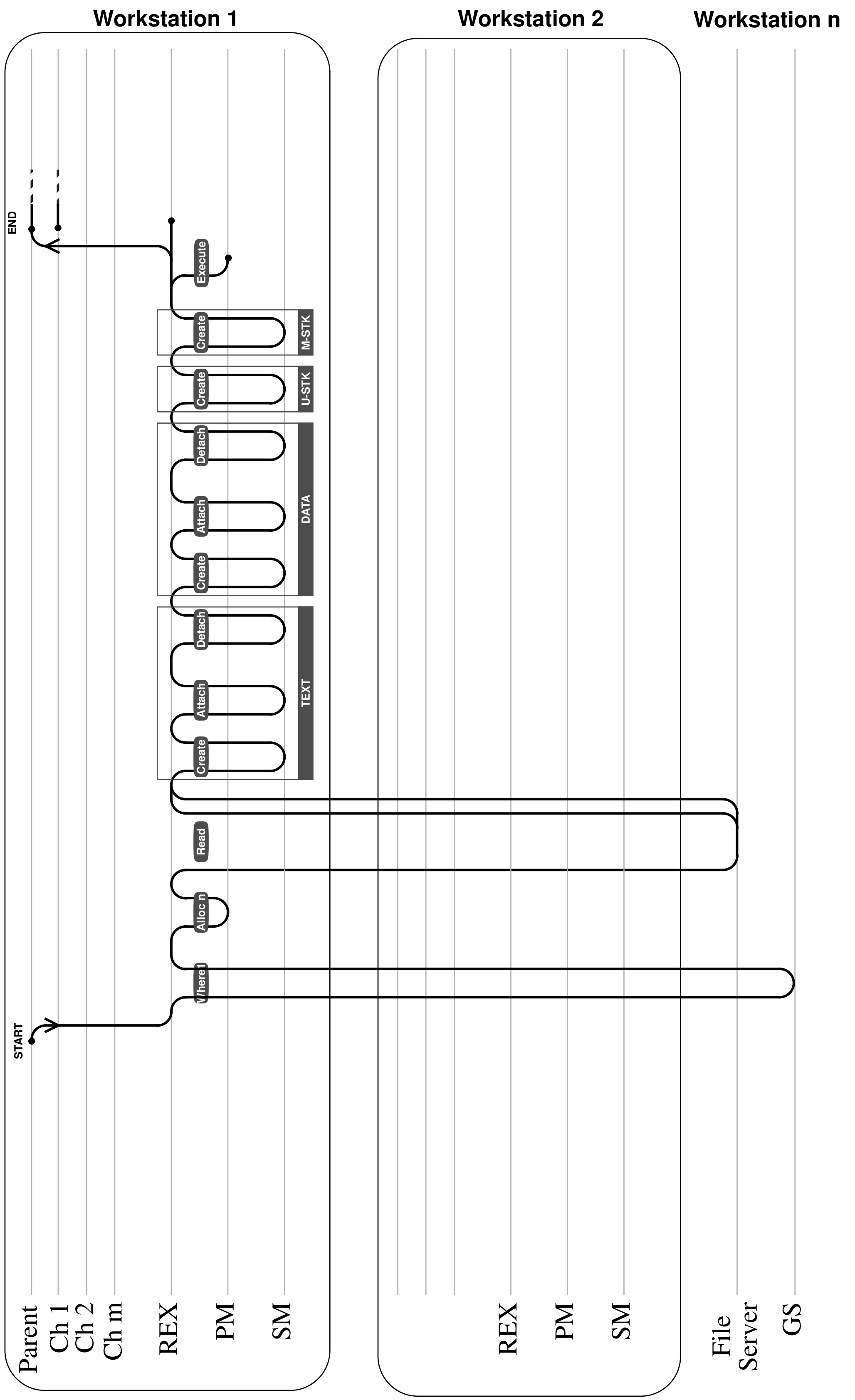

Figure 5: Order of Events in the Single Local Process Creation

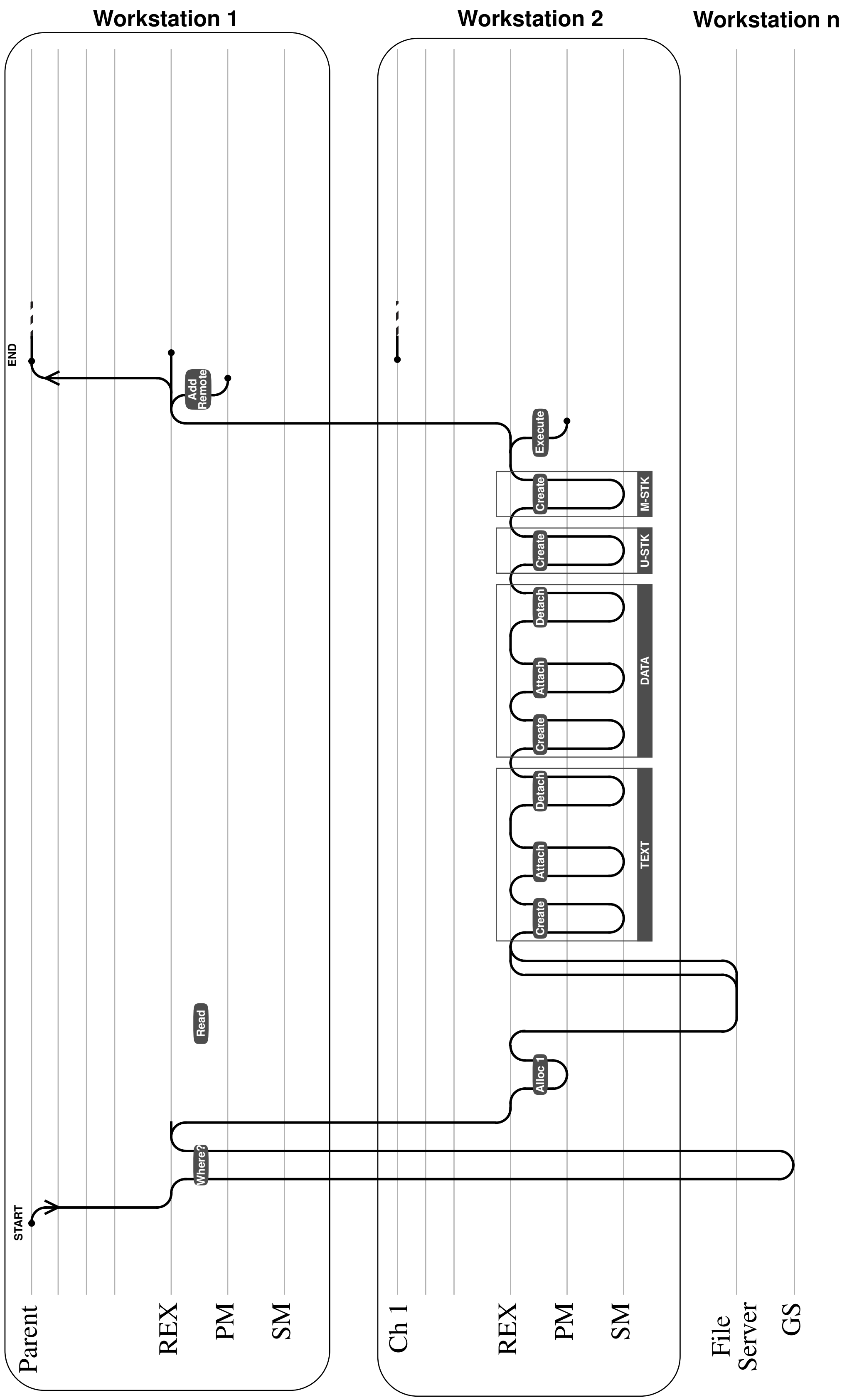

Figure 6: Order of Events in the Single Remote Process Creation

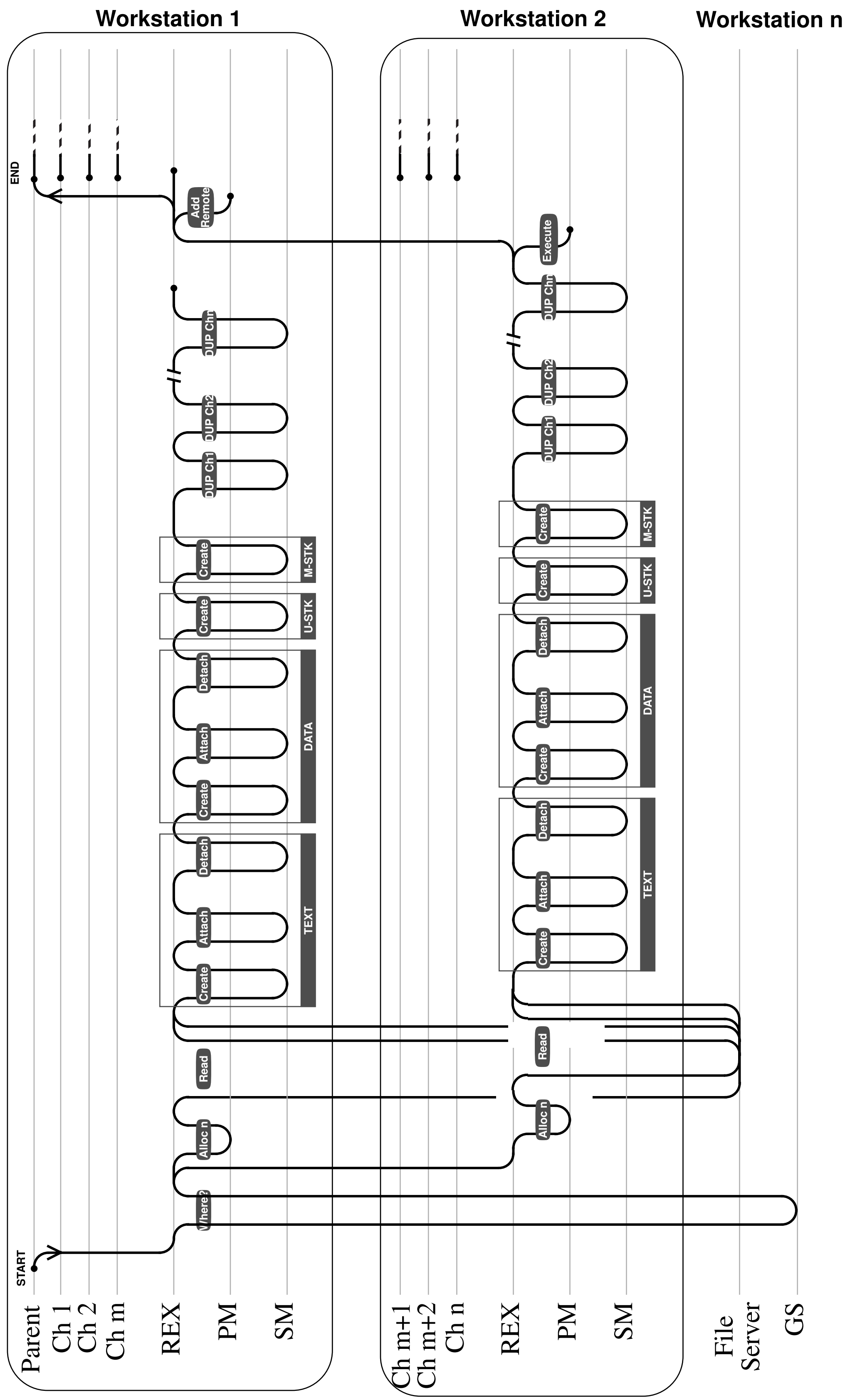

Figure 7: Order of Events in the Remote Multiple Process Creation

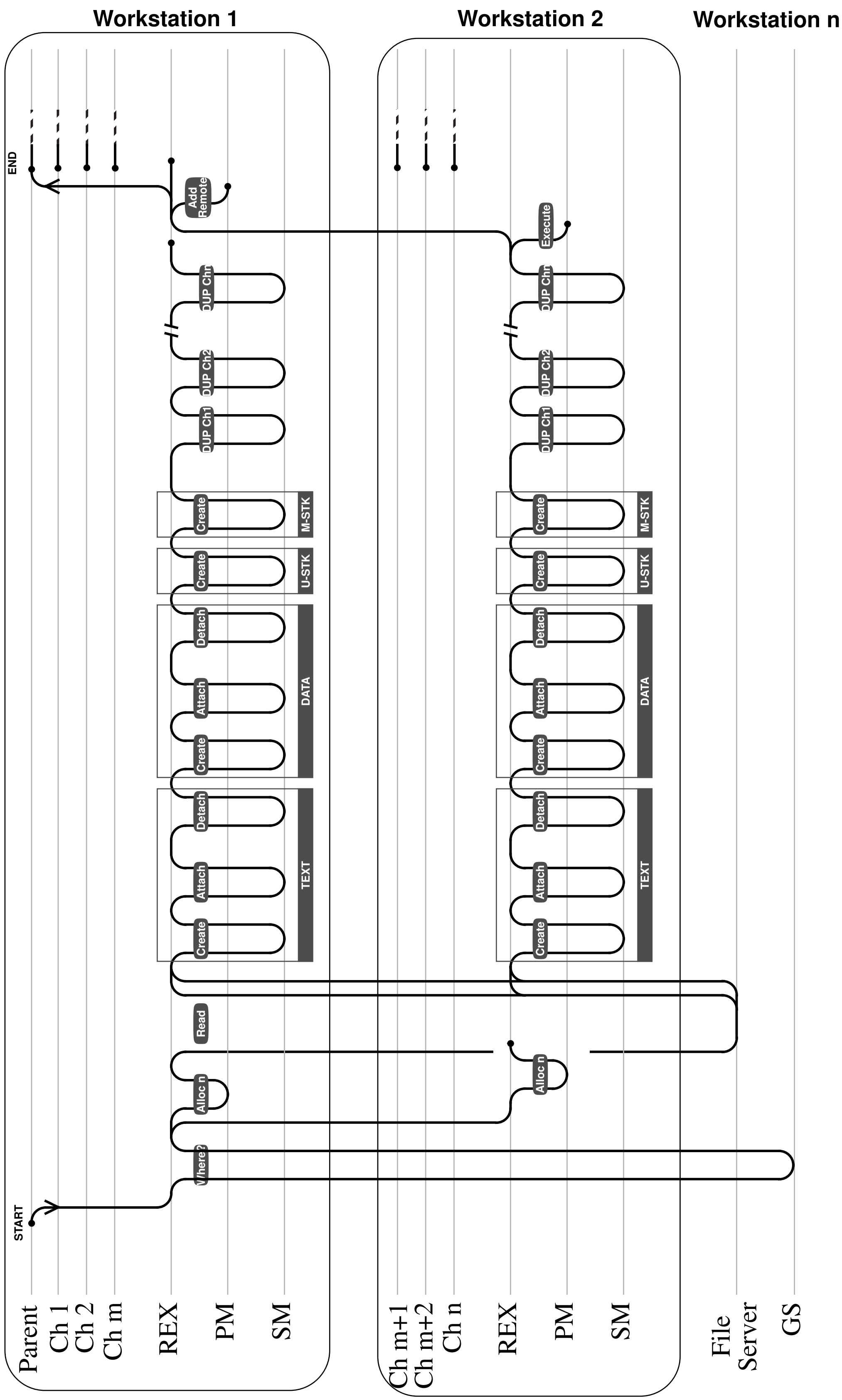

Figure 8: Order of Events in the Group Process Creation

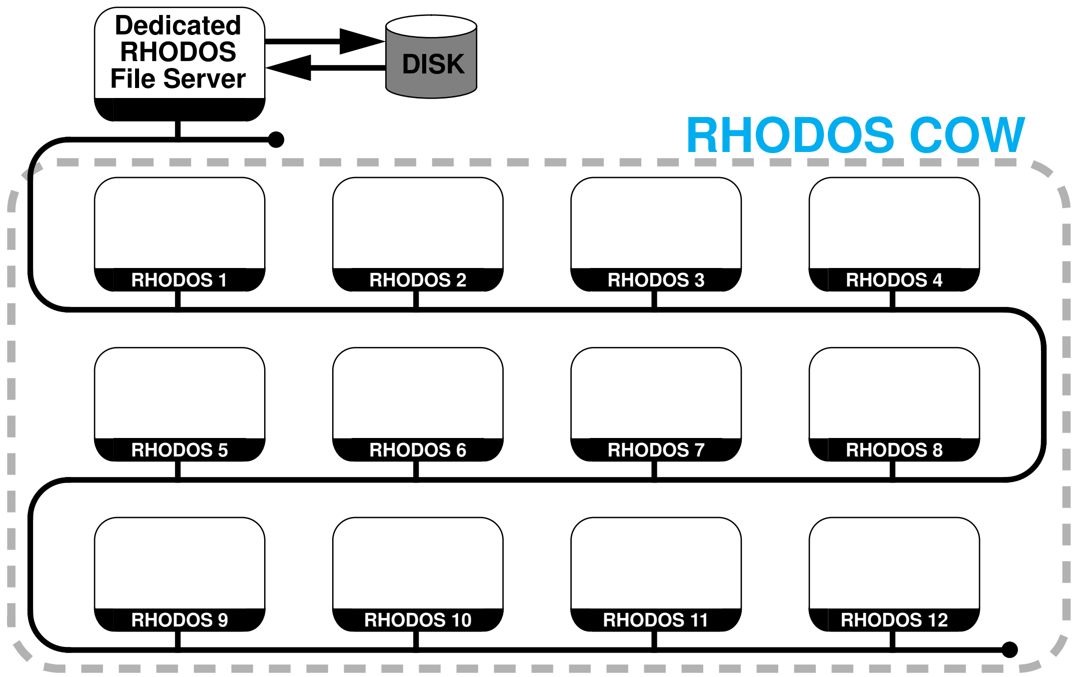

Figure 9: The RHODOS COW

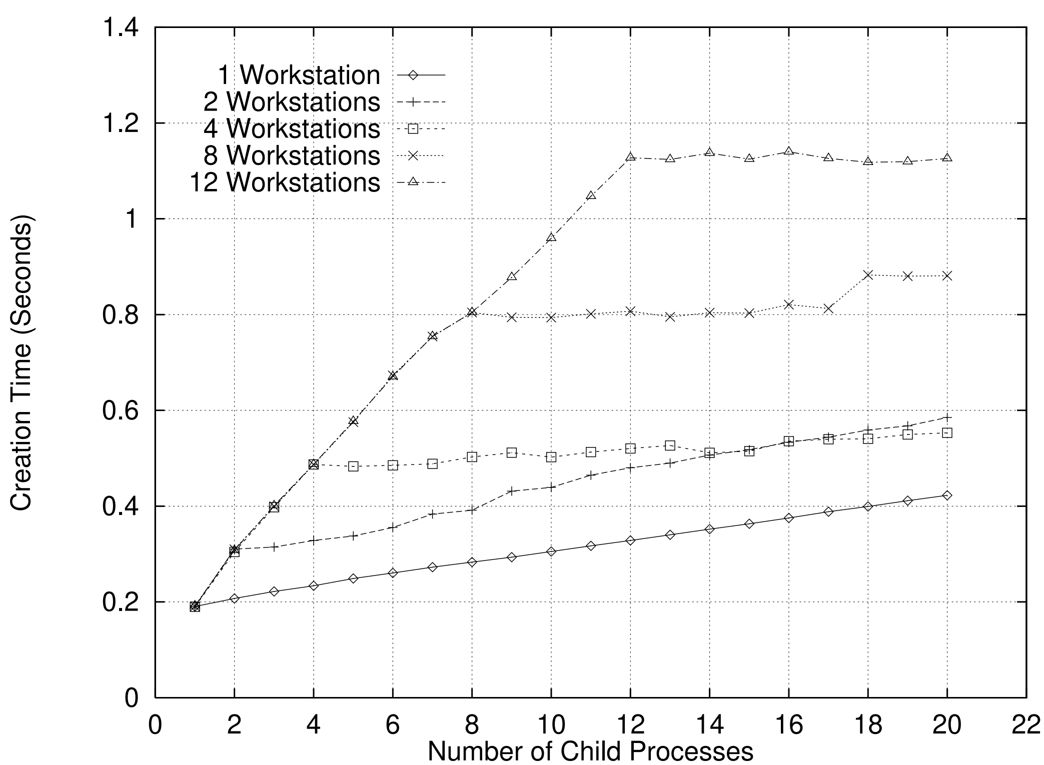

Figure 11: Multiple Process Creation

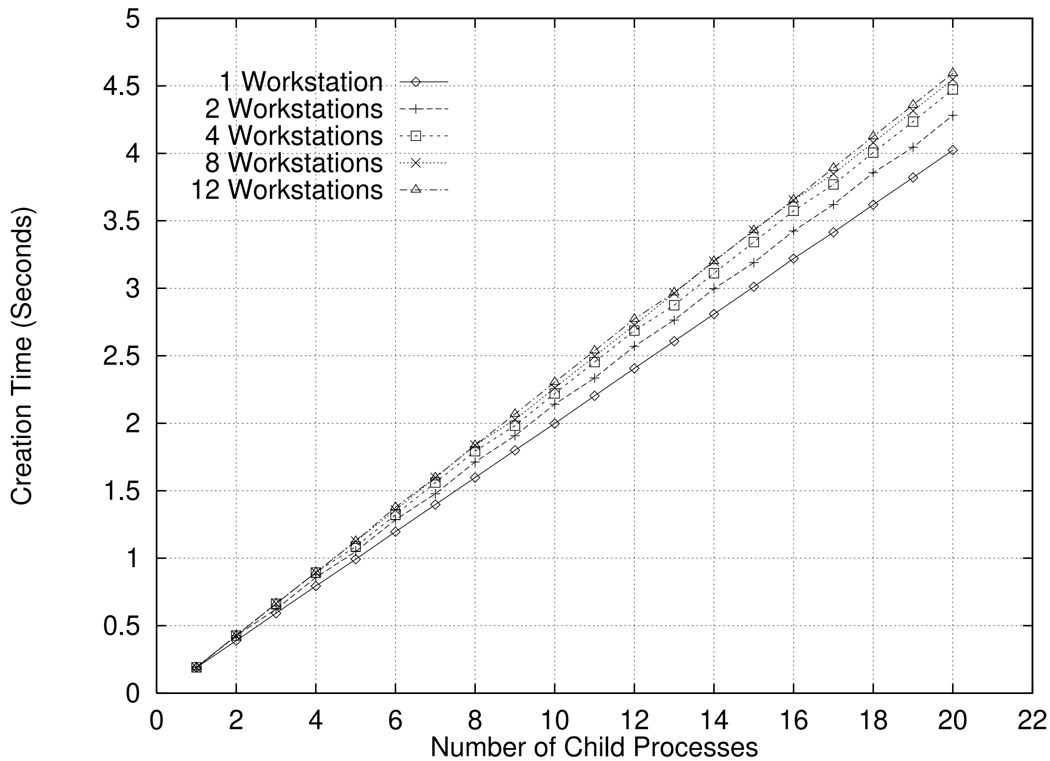

Figure 10: Single Process Creation

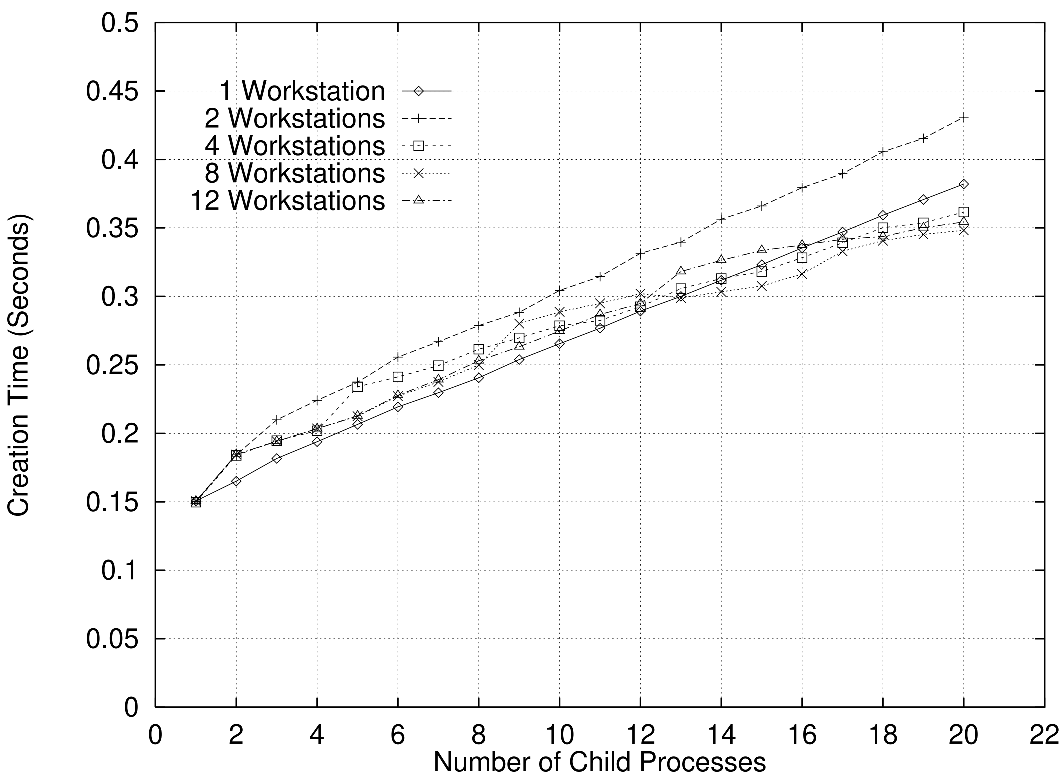

Figure 12: Group Process Creation