A GENERIC PROCESS MIGRATION ALGORITHM

Amirreza Zarrabi

Department of Computer and Communication Systems Engineering, Faculty of Engineering, University Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
gs23830@mutiara.upm.edu.my

ABSTRACT

Process migration has been advocated as a means of improving multicomputer configuration performance. The optimized migration algorithm utilized in migration event has direct effect on the efficiency and deployment of the process migration system. However, every design has preference factors results in concentration on specific aspect of the migration algorithm. There is no generic migration algorithm which could satisfy all circumstances with almost reasonable costs. This paper reviews the major issues which constitute the developer concerns when implementing a process migration algorithm. This examination indicates the existence of similarity in all process migration algorithms. A new migration algorithm is given and compared to the other algorithms. This algorithm attempts to integrate the significant features of the existing algorithms to form a generic algorithm.

KEYWORDS

Migration Algorithms, Generic Migration

1. INTRODUCTION

Process migration is the act of transferring a process between the source and the destination machines during its execution. A process has an address space containing program code and data, in addition to the states specific to underlying Operating System (OS). These states include processor execution context, open files, signal handlers, accounting information, and more [11].

Whenever transferring a process from the source machine to the destination machine, some steps should be taken, which are similar in all implementations independent to the type of migration algorithm and the underlying OS [8]:

1. The execution of the process on the source machine is suspended.
2. The state of the process is transmitted from the source to the destination machine.
3. The state of the process is reconstructed on the destination machine.
4. The execution of the process is resumed on the destination machine.
5. Information about the process is removed from the source machine.

However, the exact description on implementation of each step is specified by individual algorithms and specifications of the OSs. There are various type of process migration algorithms which differ in the process state transfer order, amount, and the period that the process is suspended [9]. Consequently, each of these algorithms would have unique characteristics that make them ideal for specific applications. Any process migration system would deploy one of the migration algorithms based on their design priority, e.g. for an OS which is not aimed at hard real-time applications, the migration delay is not an overriding concern, especially not at the cost of the increasing system complexity.

In this paper different migration algorithms are reviewed to obtain clear insight on the issues that developers try to level out by introducing the respective algorithm. A new migration algorithm is proposed which exploit significant features from all of the migration algorithms. This is a generic migration algorithm which could satisfy all circumstances with almost reasonable costs. The remainder of the paper is organized as follows: Section 2, 3 describe the basic and compound process migration algorithms. Section 4 presents the proposed migration algorithm. The qualitative evaluation of this algorithm is discussed in section 5. Section 6 is devoted to import further augmentations to the proposed migration algorithm. The simulation results are provided in section 7. In section 8 the use of migration algorithms on the virtualization environment is briefed. Finally, we present some concluding remarks.

2. Basic Process Migration Algorithms

There are four basic process migration algorithms, including: Total copy, Pre-copy, Lazy copy, and Flushing. Generally, the address space of a process constitutes the largest unit of the process states. All of the process migration algorithms are fundamentally differ in the strategy deployed to transfer the process address space while other process states are compulsory and should be present at the time of resuming the process execution on the destination machine.

Irrespective to the strategy, a process would be in four different states while a migration event is in progress:

Normal→Pre-migration→Migration→Post-migration→Normal

A process would be executed in normal state until a request for migration is produced. The pre-/post-migration states are deployed to compensate the overhead of transferring the process address space or eliminating the residual dependencies on the source machine. In the rest of this section, we provide a brief discussion on advantages and disadvantages of different algorithms.

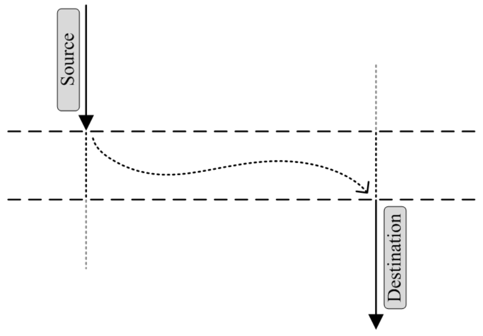

Figure 1. Total Copy Algorithm.

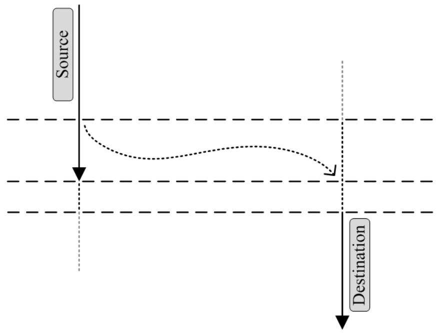

Figure 2. Pre-copy Algorithm.

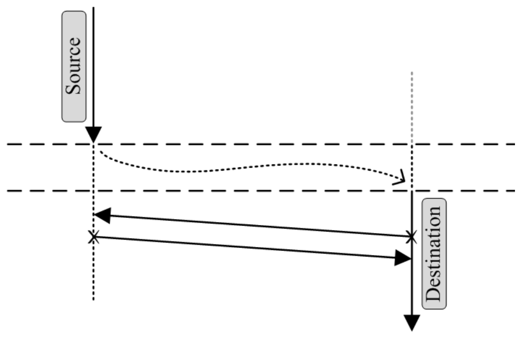

Figure 3. Lazy Copy Algorithm.

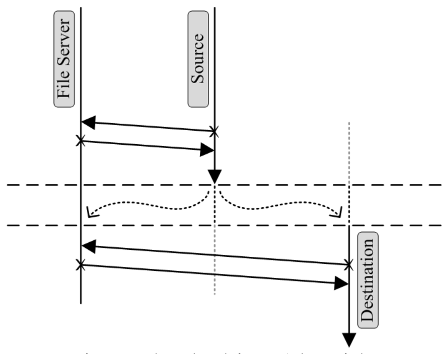

Figure 4. Flushing Algorithm.

2.1. Total Copy Algorithm

It is the earliest and most frequently used process migration algorithm. This algorithm transfers the entire state of the process in the migration event before the process execution resumed on the destination machine. In spite of imposing a lengthy delay in the process execution time, its conceptual simplicity allows straightforward implementation. Figure 1 illustrates the total copy algorithm. Charlotte [1] and Amoeba [10] are example OSs that uses this algorithm.

There is a variant of this algorithm, which attempts to minimize the overhead of transferring of process address space by limiting the migration to the dirty pages. All clean pages in the process address space are transferred on request from the local backing store, but it instructs the destination machine to have a backing store which has exact duplication of the source machine contents.

The process would alternate between two states:

Normal →Migration→Normal

2.2. Pre-copy Algorithm

This algorithm operates almost similar to the total copy algorithm, but it would transfer the address space of the process while the process continues its execution in parallel. Ultimately, the execution of the process on the source machine would be suspended and all other process states, including pages in the process address space which were modified following the initial transfer would be sent to the destination machine. Its time flow is displayed in Figure 2. V [12] is the first OS that uses this algorithm.

This algorithm has been proposed to reduce the migration delay introduced in the process execution time by the total copy algorithm. However, it would result in multiple state transfers of the process as depending on the exact memory activity pattern of the process some pages would be modified after being transferred.

The process would alternate between three states:

Normal→Pre-migration→Migration→Normal

2.3. Lazy Copy Algorithm

This algorithm would transfer only the minimum necessary information for the process to resume its execution on the destination machine. This would typically consist of the process state other than pages in the address space. When a reference is made to the page which still resides on the source machine, page fault would occur and a request is generated to transfer the missing page to the destination machine from the source machine. It is demonstrated in Figure 3. Accent [13] and Mach [6] are sample OSs that implement this algorithm.

This algorithm could substantially reduce the migration delay as the minimal information is transferred before the process execution resumed on the destination machine. Moreover, there is a possibility of an overall reduction in the network traffic which would be resulted from avoiding the need to transfer all process state information. Nevertheless, in this algorithm, the state of the process would be distributed between the source and destination machines results in residual dependencies on the source machine.

The process would alternate between two states:

Normal →Migration→Post-migration→ Normal

The process would never returns to its normal state until it ends or returns to the source machine.

2.4. Flushing Algorithm

This algorithm consists of a third entity which plays as a global backing store which is accessible by both source and destination machines. The flushing algorithm depends on the OS implementation of virtual memory and only feasible if it utilizes ordinary files for the virtual memory backing store.

While migrating a process the dirty pages in the process address space would be transferred to the global backing store, all other states are sent to the destination machine, and then process execution resumed. The process address space pages are retrieved from the global backing store on demand.

The flushing algorithm trades off the increased overhead of the flush operation for the elimination of residual dependencies that exists in the lazy copy algorithm. This algorithm is depicted in Figure 4. This algorithm is initially introduced in Sprite [3] OS.

The process would alternate between two states:

Normal →Migration→Normal

3. COMPOUND PROCESS MIGRATION ALGORITHMS

Other algorithms could be generating by combining different basic algorithms. Among all of the possible combination, the post-copy algorithm is receiving good attention from the developer community. There was a recent implementation of post-copy algorithm under openMosix [5]. Moreover, the thorough discussion in [4] proposed the utilization of the post-copy algorithm for the live migration of virtual machines.

The post-copy algorithm is a combination of the pre-copy and lazy copy algorithms. It is similar to the pre-copy algorithm as the process states are transferred while the process is executing. This algorithm tries to avoid residual dependencies by moving the process address space as it is executing on the destination machine.

As with the lazy copy algorithm, the process may refer to pages which are still reside on the source machine. Therefore, a request is generated to transfer the missing page which takes precedence over the normal page transfers. This algorithm is illustrated in Figure 5.

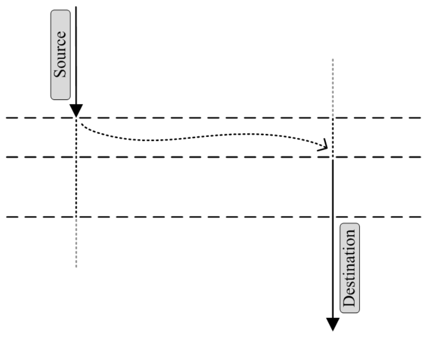

Figure 5. Post-copy Algorithm.

The process would alternate between three states:

Normal →Migration →Post-migration → Normal

The assisted post-copy algorithm is proposed in [7]. It is a synthesis of the pre-copy and post-copy algorithms which considered to be the only process migration algorithm that utilized both pre-migration and post-migration states to optimized the migration event.

In this algorithm, the process would alternate between four different states:

Normal →Pre-migration →Migration →Post-migration →Normal

4. GENERIC PROCESS MIGRATION ALGORITHM

The proposed process migration algorithm is a combination of pre-copy, post-copy, and flushing algorithms. It consists of three cycles, referring to as pre-migration, migration, and post-migration cycles as demonstrated in Figure 6.

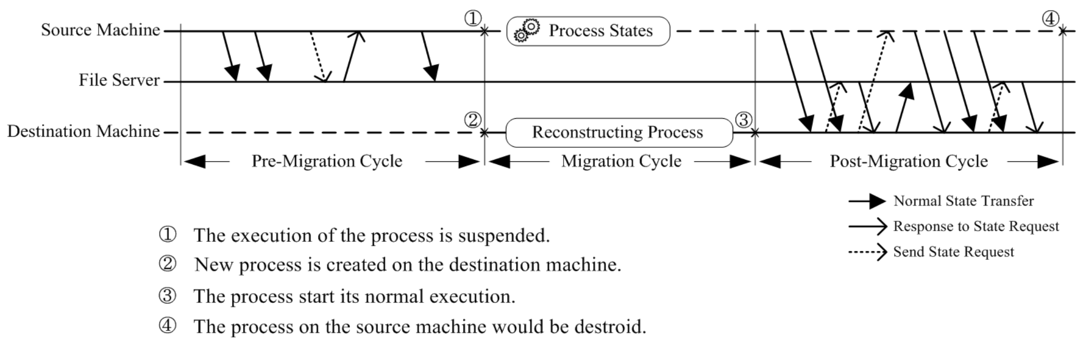
Source Machine
File Server
Destination Machine
①
②
③
④
Process States
Reconstructing Process
Pre-Migration Cycle
Migration Cycle
Post-Migration Cycle
Normal State Transfer
Response to State Request
Send State Request
① The execution of the process is suspended.
② New process is created on the destination machine.
③ The process start its normal execution.
④ The process on the source machine would be destroid.

Figure 6. Proposed Migration Algorithm Cycles.

In the pre-migration cycle, the dirty pages from the process address space would be transferred to the file server as they are being modified by the process. We restrict this operation to regions of the process address space which map a file, and the modification to the file should be apparent to all processes retrieving its content simultaneously.

When entering to the migration cycle, two processes would exist in the network which both represent the migrating process. The process states other than pages in the address space of the process would be transferred from the source machine to the destination machine. Afterward, the process at the destination machine would be resumed as a concluding step.

In the post-migration cycle, the dirty pages from the address space of the process which their contents are private for the process are transferred to the destination machine while all other pages would be obtained from the local backing store. Meanwhile, the file server would act similar to the pre-migration cycle.

The post-migration cycle would be transformed to the pre-migration cycle when all private pages moved to the destination machine so that the process can be migrated to another machine or even migrate back to the source machine.

According to the description, the process would alternate between three different states:

Pre-migration→Migration→Post-migration→Pre-migration.

5. QUALITATIVE EVALUATION

Quantitative evaluation of a migration algorithm is very difficult as this requires the algorithms to be implemented and tested on similar hardware, network, and process environment. However, qualitative comparison between the proposed migration algorithm and the others would give some indication of its potential value. Some questions should be answered, including [8]:

How long is the migration delay? The delay of the proposed migration algorithm is as short as that provided by the lazy copy algorithm since the migration cycle would only transfer the minimal process states to resume its execution on the destination machine. Other algorithms require transfer of additional states, and it is rational to conclude that they produce longer delay.

How much is the migration overhead on the process run-time? The migrated process may experience a delay in post-migration cycle due to the requests for pages before they are transferred in their normal order. The process run-time would be slightly longer than the pre-copy algorithm assuming a minimum duplication in its concluding state transfer. However, process run-time may be shorter than the lazy copy algorithm as some pages in process address space could be transferred before the process references them. The proposed migration algorithm would transfer entire state of the process with some degree of parallelism, reducing the process run-time compared to the total copy and flushing algorithms.

Is there any dependency on the source machine? Similar to the total copy, pre-copy, and flushing algorithms, the proposed migration algorithm does not produce any source machine dependency. Nevertheless, it attempts to reduce the migration delay by simulating lazy copy algorithm. The temporary source machine dependency would be eliminated after transferring the remaining process states to the destination machine.

How much is the network traffic? The proposed migration algorithm results in higher network traffic than other migration algorithms. This is fairly obvious as some pages in the process address space are transferred to the file server in the pre-migration cycle as the process executing even if it does not intend to migrate. Neglecting this overhead, the network traffic produced by the proposed migration algorithm is almost equal to the total copy, pre-copy, and flushing algorithm.

6. Further Augmentation

The resumption of process execution would be accompanied by a page fault which requests a page from code segment of the process address space. Other pages in the process address space would be transferred either in their normal order or using page faults. Therefore, the processes are often executed in suboptimal performance because of the time consumed for page faults. The migration cycle of proposed migration algorithm can be tweaked to transfer the currently needed pages from the code and stack segments in advance.

Prefetching techniques could be deployed in the post-migration cycle of the proposed migration algorithm. Generally, in existing prefetching techniques requests are issued by a data consumer to fetch data before the actual access takes place. Additionally, the prefetching technique could be implemented in push mode to decrease the network traffic of page requests round-trip. In the push mode, the source machine is responsible to determine which page to prefetch and transfer. An aggressive prefetching theme is used, which constantly attempts to push the next missing page at the earliest opportunity by considering the pages following the last requested page as a working set [2]. It is reasonable as the source machine only has partial information for process memory references, which is acquired by page requests from the destination machine.

7. Simulation Results

Different process migration algorithms have been implemented on various OSs, which make a direct compression impossible to obtain, e.g. OS specification have direct effect on performance of similar algorithms implemented on different OSs. Therefore, analysis of the migration algorithms are conducted by simulating migration events in multiple user space processes as a method to compare their efficiency.

As the process address space constitute the largest overhead in migration events, the transfer of the address space could be considered as a measure to evaluate the whole migration event. The simulated event represents the migration event of a process with 10MB of address space which randomly access/modify pages in 100 Mbps network.

The pre-copy algorithm is the only migration algorithm which exploits the process pre-migration state as shown in Table 1. This time is constant irrespective to the number of dirty pages in the address space of the process. However, it would adversely affect the migration delay. The migration delay for the pre-copy algorithm in the best case when all pages are clean is larger that the expected value. This is because of transmission delay of the pre-migration state. Similarly, this effect is apparent for the worst case where all page should be transferred in the migration state in Table 2.

Other pages in the process address space would be transferred either in their normal order or using page faults. Therefore, the processes are often executed in suboptimal performance because of the time consumed for page faults. The migration cycle of proposed migration algorithm can be tweaked to transfer the currently needed pages from the code and stack segments in advance. The processor instruction pointer and stack pointer registers could be exploited to identify the intended code and stack page respectively.

Table 1. Process in Pre-migration State.

	Normal Case	**Best Case**	**Worst Case**
Total Copy	0	0	0
Pre-copy	776ms	776ms	776ms
Post -copy	0	0	0
Lazy Copy	0	0	0

Table 2. Process in Migration state.

	Normal Case	**Best Case**	**Worst Case**
Total Copy	780ms	780ms	780ms
Pre-copy	202.5ms	52.5ms	828ms
Post -copy	4.5ms	4.5ms	4.5ms
Lazy Copy	4.5ms	4.5ms	4.5ms

Table 3. Process in Post-migration State.

	Normal Case	**Best Case**	**Worst Case**
Total Copy	0	0	0
Pre-copy	0	0	0
Post -copy	900ms	778ms	1300ms
Lazy Copy	∞	∞	∞

The post-copy and lazy copy algorithms have the lowest migration delay as the minimum process sate is transferred for the migration event. In the best case for post-copy algorithm when no page fault is generated, the process would remain in post-migration state almost equal to the migration state of total copy algorithm in Table 3. However, the worst case would be increased to 1300ms because of the over head of post-copy algorithm computation.

The generic migration algorithm presents an arbitrary behaviour in different situations. It is compared to other algorithms in Figure 7. The generic migration algorithm would overlap with all other migration algorithms. Therefore, the migration algorithm would behave similar to different migration algorithms in various situations, e.g. in the worst case it would behave similar to the post-copy algorithm.

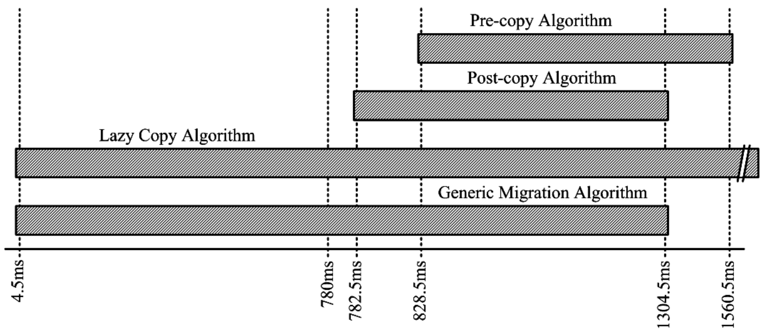

Figure 7. Duration of the Migration Events.

8. RELATED WORKS

Even though in this paper we concentrate on the migration algorithms in the context of transferring the processes, similar algorithms could be deployed in the virtualization environments for the live migration of Virtual Machines (VMs). Most of the live migration techniques use the pre-copy approach for implementation of the migration facility. The popular migration system, named VMotion, which has been shipping since 2003 as an integral part of the VMware VirtualCenter product is designed based on pre-copy algorithm [7]. This algorithm is adopted so that the memory of the VM is transferred to the destination while the VM is running in the pre-migration state. Moreover, a viable post-copy technique for live migration of VMs is described in [4]. The generic migration algorithm could be adopted for transferring VMs in the network as basically it shows the behaviour of the post- and pre-copy migration algorithms.

9. CONCLUSIONS

In this paper we presented a generic process migration algorithm which benefits from significant features of the existing algorithms. We were concerned with the design of the migration algorithm which could behave similar to the basic migration algorithms. According to the type of pages in the process address space, the general migration algorithm would incline to the particular algorithm attributes.

REFERENCES

[1] Y. Artsy and R. Finkel. Designing a process migration facility: The charlotte experience. *Computer*, 22(9):47–56, 1989.

[2] P. Cao, E.W. Felten, A.R. Karlin, and K. Li. *A study of integrated prefetching and caching strategies*, volume 23. ACM, 1995.

[3] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software Practice and Experience*, 21(8), 1998.

[4] M.R. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[5] R.S.C. Ho, C.L. Wang, and FC Lau. Lightweight process migration and memory prefetching in openmosix. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE, 2008.

[6] D.S. Milojicic, W. Zint, A. Dangel, and P. Giese. Task migration on the top of the mach microkernel. In *Mobility*, pages 134–151. ACM Press/Addison-Wesley Publishing Co., 1999.

[7] M. Nelson, B.H. Lim, G. Hutchins, et al. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 25–25, 2005.

[8] M. Noack. Comparative evaluation of process migration algorithms. *Master's thesis, Dresden University of Technology–Operating Systems Group*, 2003.

[9] M. Richmond and M. Hitchens. A new process migration algorithm. *ACM SIGOPS Operating Systems Review*, 31(1):31–42, 1997.

[10] E.T. Roush and R.H. Campbell. Fast dynamic process migration. In *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*, pages 637–645. IEEE, 1996.

[11] E. Steketee, W.P. Zhu, and P. Moseley. Implementation of process migration in amoeba. In *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*, pages 194–201. IEEE, 1994.

[12] A.S. Tanenbaum and A.S. Woodhull. *Operating systems: design and implementation.* Alternative eText Formats Series. Pearson Prentice Hall, 2006.

[13] M.M. Theimer, K.A. Lantz, and D.R. Cheriton. *Preemptable remote execution facilities for the V-system,* volume 19. ACM, 1985.

[14] E. Zayas. Attacking the process migration bottleneck. In *ACM SIGOPS Operating Systems Review*, volume 21, pages 13–24. ACM, 1987.

Authors

Amirreza Zarrabi has received his master's degree in Computer Communication and Networks from University Putra Malaysia in 2012. His research interests are in computer architecture, device modelling, embedded systems, operating systems, system security, web services, and distributed computing. During his thesis, he was working on distributed memory, file management, and process migration in Linux operating system. Currently he is working on robust transport protocol for dynamic high-speed networks as a research engineer in an international investment solution provider company in Kuala Lumpur