An Adaptive Pre-copy Strategy for Virtual Machine Live Migration

Ching-Hsien Hsu[1,2], Sheng-Ju Peng[1], Tzu-Yi Chan[1], Kenn Slagter[3], and Yeh-Ching Chung[3]

[1] Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu, 30012 Taiwan
{chh,m10102022,b10002099}@chu.edu.tw
[2] School of Computer and Communication Engineering, Tianjin University of Technology, Tianjin, 300384 China
[3] Department of Computer Science, National Tsing Hua University, Hsinchu, 30013 Taiwan
{kennslagter,ychung}@cs.nthu.edu.tw

Abstract. Live migration technology for virtual machines provides greater flexibility when scheduling tasks in a cloud environment. This flexibility helps increase the utilization of resources within the cloud. A key component of live migration technology is the pre-copy strategy. The pre-copy strategy allows virtual machine to perform live migration without interruption of service. However, in order for live migration to be efficient, it is imperative that virtual machines' memories are not limited by bandwidth and that the downtime of the virtual machines involved is minimal.

This paper presents an adaptive pre-copy strategy for virtual machine live migration called Multi-Phase Pre-Copy (MPP). In iterative pre-copy stage MPP transmits memory pages only when a predefined threshold is met. This strategy significantly reduces unnecessary migration of memory pages.

Keywords: Virtualization, virtual machine, live migration, multi-phase pre-copy.

1 Introduction

Investment by businesses and governments has led to a rapid growth in cloud computing research. Cloud computing applications use a combination of networked computing resources and virtualization techniques and architecture. Virtualization is achieved by having physical machines (called nodes) in the network run virtual machines. These virtual machines can be divided up in different ways and can be asked to perform different tasks.

Virtualization is a key technology in cloud computing. Consequently, effective management of virtual machines and related technologies is required in order to make efficient use of resources on the cloud. One of the ways to make efficient use of the underlying hardware on the cloud is to dynamically transfer virtual machines throughout the network in order to make better use of the resources on the network. Dynamic transfer of virtual machines over the network is known as migration.

Live migration technology is based on prevalent virtual machine migration technology. Previous migration techniques required a short downtime of a system whenever a virtual machine gets transferred over the network. The purpose of live migration is to allow virtual machines to migrate elsewhere on the network without any perceivable interruption of a service by the user.

Live migration of virtual machines can be divided into pre-copy and post-copy methods. The most common live migration method is pre-copy. With the pre-copy method a hypervisor copies all the memory pages from the source node to the destination node while the virtual machine is running on the source node. With the post-copy method the virtual machine is paused at the source, and the execution state of the virtual machine is transferred to the target. Once the virtual machine is transferred to the target the virtual machine resumes execution and the source pushes stored memory pages to the target via a technique known as pre-paging.

Migration time for post-copy migration is relatively low compared to pre-copy migration, but the overall downtime of the virtual machine in post-copy is longer than it is for pre-copy. Therefore, this paper focuses on improving pre-copy migration. The Xen hypervisor [1][2] is a well-known open source hypervisor used by industry and academia that implements pre-copy migration. For these reasons, this paper focuses on reducing downtime and total migration time when doing pre-copy migration by the Xen hypervisor.

A drawback of the traditional Xen pre-copy migration process is that it repeatedly copies memory pages and processor (CPU) states across the network. The copying of memory page occurs when the data in the memory page changes, and becomes a dirty page. Xen will send this dirty page to the target even if the dirty page remains unchanged from the last time it was copied. Consequently, this prolongs the overall migration time.

In this paper, we propose a new multi-phase pre-copy method (MPP) that reduces the number of pages being shipped over the network, and reduces overall migration time. MPP transfers pages over the network based on a set of threshold values which determine if a page should be transferred to its target or not.

The rest of this paper is organized as follows. In Section 2 we discuss related work. Section 3 presents some background. In Section 4, we present our proposed technique. In Section 5, we present our performance evaluation. Finally, the conclusion and future work are presented in Section 6.

2 Related Work

Virtual Machine Live Migration (VMLM) has become a hot topic in both academia and industry. Topics on VMLM can be found in both practical applications as well as research papers. Traditionally VMLM migrates the internal memory state, and importantly involves memory image transfer. Some memory transfer research focuses on handling the state of the virtual machine memory [3][4][5]. Additional research by [5][6][7] provides a virtual machine management platform that provides an uninterrupted service when migrating virtual machines.

Pre-copy algorithms [5][7] have been presented that enhance pre-copy efficiency for when machine reads and writes are numerous. These algorithms focus on alleviating the amount of downtime that occurs due to the frequent number of dirty pages created as a consequence. From a different perspective there is also research [8] that looks into how bandwidth can be dynamically adjusted during VMLM in order to limit network traffic.

To further improve the efficiency of the pre-copy algorithm [6] introduces time-series prediction techniques. Time-series prediction is done by updating records and using statistics each time a dirty page is sent from the source to the target. However, the associated formula expects parameters used to define a page to be static even though these parameters can easily change. Furthermore, even though the dirty page threshold in its formula can significantly reduce pre-copy transmission times in iterative rounds, it increases the downtime.

3 Background

Virtual machine migration has become an essential feature in cloud computing. There are various reasons why virtual machine migration may be instigated including: hardware maintenance, lack of resources, and load imbalance. Despite improved or more stable performance after migration, it is best if there is no perceivable interruption of service during migration by the user.

Virtual machine migration can be divided into two distinct categories: offline migration and online migration. Online migration itself can be divided into two distinct subcategories cold migration and live migration. Details of these two subcategories are described below.

According to [9] when one migrates virtual machines on a virtual system one needs to consider downtime and total migration time. Downtime is the time it takes a source host to suspend a virtual machine until the time the virtual machine resumes on the target host. In other words downtime refers to the time when the virtual machine is unresponsive. Total migration time is the time from the start of the migration process on the source host until the end of the migration process, which is after the time the virtual machine resumes on the target host and once the source virtual machine has been deactivated and discarded.

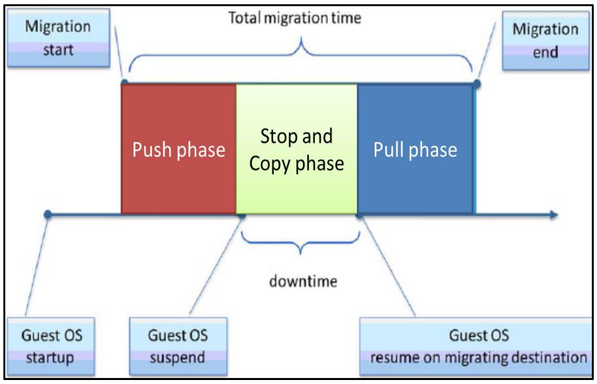


Fig. 1. Virtual Machine Migration Phase Diagram [4]

Figure 1 shows the three different phases that occur during live migration [9], and the relationship they have between each other and their effect on downtime and total migration time. The phases that occur during virtual machine migration are:

1. *Push phase*: The source virtual machine continues to operate while content is moved to the target.
2. *Stop-and-copy phase*: the source virtual machine stops executing. Any remaining content found at the source is removed, following the move, the migrated virtual machine on the target begins to operate.
3. *Pull phase*: a virtual machine system is up and running on the target host during migration. Any content missing on the target is taken (pulled) from the source.

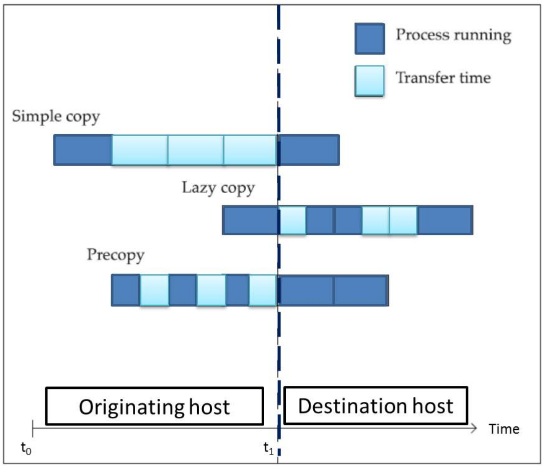


Fig. 2. The three ways migration is performed [4]

In general, there are three common ways that migration is performed [10]:

1. *Simple copy*: also known as “pure stop-and-copy” [11] halts the virtual machine at the source, copies memory pages from the source to the target, and then starts a new virtual machine on the target. The advantage of this approach is that it is the simplest one to implement. However, the disadvantage of this approach is that there must be a significant amount of downtime and overall migration time. How long it takes to migrate depends on the size of the memory used by the virtual machines.
2. *Lazy copy*: also known as “pure demand-migration”[12] uses a short stop-and-copy phase to transfer over any vital kernel data to the target. The target virtual machine is then started. Memory pages are then transferred over the network when used for the first time due as a consequence of a page fault. The advantage of this approach is that it has a much shorter downtime compared to the simple copy method. However, the total migration time is lengthened. Furthermore, the performance of the target virtual machine after migration will likely be poor until a substantial number of pages have been transferred across.

3. *Pre-copy*: balances the concerns found in simple copy and lazy copy methods. It combines an iterative push phase with a typically short stop-and-copy phase. During the iterative push phase pages are transferred in rounds to the virtual machine ready for them to be of use in the subsequent round that needs them. Since every virtual machine has a set of pages that are used often and are a poor candidate for the pre-copy method the number of rounds is bounded based on analysis of a writing working set.

Pre-copy live migration operation can be divided into six stages. Each stage represents a different mode of operation.

1. *Stage 0 Pre-migration*: Prior to migration a host with sufficient resources is selected.
2. *Stage 1 Reservation*: A request is made to migrate from the source to the target host.
3. *Stage 2 Iterative Pre-Copy*: In the first iteration of the round, all source pages are transferred to the host. Any pages that change (dirty pages) are then sent in subsequent rounds.
4. *Stage 3 Stop and Copy*: Stops the virtual machine on the source machine and redirects network traffic to the target machine. CPU state and any dirty memory pages get transferred as well. Both source and target now have a suspended copy of the virtual machine. The source virtual machine will resume if there is a failure.
5. *Stage 4 Commitment*: Target host informs the source host that it has received the complete virtual machine image. The source host can now discard the old virtual machine.
6. *Stage 5 Activation*: The virtual machine will now run as normal on the host machine.

Xen's iterative pre-copy stage categorizes used memory pages into three types of bitmaps. These bitmaps are described as follows:

1. *To_Send*: marks the pages which were dirty on the previous round, this identifies which pages need to be transmitted for the current iteration.
2. *To_Skip*: marks which pages can be ignored for the current iteration.
3. *To_Fix*: marks which pages should have been transmitted in the previous iteration.

These bitmaps represent those memory pages that need to be sent from the source host to the target host. In the first round of the iterative process, the entire page of memory within the virtual machine is sent. Those memory pages that are to be sent are marked. It then determines whether pages that occur in the subsequent round of the iterative process should be sent or skipped.

Xen's iterative pre-copy stage has several termination conditions that ensure it completes the stage in a timely manner. If any of these termination conditions are satisfied the iterative pre-copy stage ends and the stop-and-copy stage begins. The pre-copy termination conditions are as follows:

1. Less than 50 pages were dirtied during the last pre-copy iteration
2. 29 pre-copy iterations have been carried out.

3. More than 3 times the total amount of memory allocated to the VM has been copied from the source host to the target host.

If the first termination condition is satisfied there will be a short downtime. This is because the final iteration will send only a small number of dirty pages from the source to the target. If the second or third condition is satisfied there will be a longer downtime. This is because there will be a forced migration, a transition to the stop-and-copy stage, and a considerable number of dirty pages transferred from the source to the target host.

The termination conditions used by Xen gives it some advantages to a naïve implementation of the iterative pre-copy process. Without these termination conditions, iterative pre-copy would take a long time to terminate. This is because memory typically has regular periods of intensive reading and writing. Furthermore, workloads typically have a small set of pages that are frequently accessed. By sending pages when memory is not undergoing intensive read and write sessions, Xen prevents many unnecessary page transfers. This in turn prevents long migration times.

4 Multi-phase Virtual Machine Migration

In this section we describe improvements made to the iterative nature of the pre-copy method. The concept presented here applies lazy pre-copy mechanisms to different stages of the pre-copy mechanism and then applies different judging criteria to improve overall efficiency during live migration.

A. Lazy Pre-Copy

Live migration, migrates virtual machines over the network from a source host to a target host. It is tempting to use available bandwidth as metric as a way to improve performance of LMVM. Unfortunately, it is impossible to predict what the bandwidth will be based on data collected from the bandwidth history. Furthermore, an incorrect prediction is likely to have a negative effect on performance.

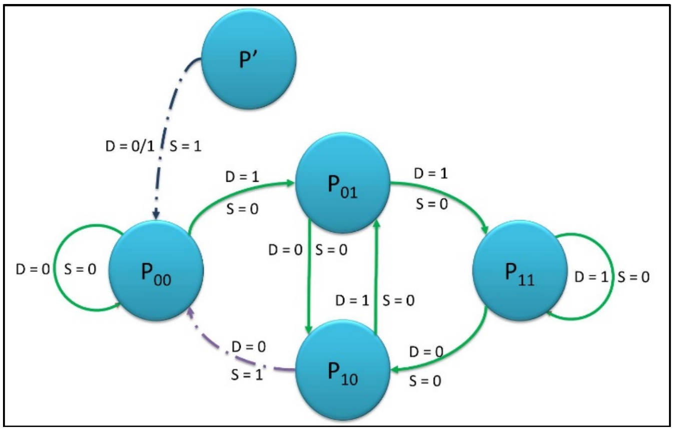


Fig. 3. Lazy Pre-Copy State Diagram

In order to improve bandwidth usage we use a lazy pre-copy mechanism. The lazy pre-copy (LPC) mechanism improves the effective bandwidth by reducing the number of memory pages transmitted over the network. A state diagram for the lazy pre-copy mechanism is presented in Figure 3. A description of the state transitions found in lazy pre-copy is as follows:

1. $P' P_{00}$: Initialize, send all memory pages (dirty or not) from source host to target host.
2. $P_{00}P_{00}$: pages are *not* dirty, pages are *not* sent from source host to destination host.
3. $P_{00}P_{01}$: pages are dirty, pages are *not* sent from source host to destination host.
4. $P_{01}P_{10}$: pages are *not* dirty, pages are *not* sent from source host to destination host.
5. $P_{01}P_{11}$: pages are dirty, pages are *not* sent from source host to destination host.
6. $P_{11}P_{11}$: pages are dirty, pages are *not* sent from source host to destination host.
7. $P_{11}P_{10}$: pages are *not* dirty, pages are *not* sent from source host to destination host.
8. $P_{10}P_{01}$: pages are dirty, pages are *not* sent from source host to destination host.
9. $P_{10}P_{00}$: pages are *not* dirty, pages are sent from source host to destination host.

Lazy pre-copy improves the migration process by using bandwidth more efficiently. Lazy pre-copy achieves this by waiting an extra round before it transfers data, this reduces the number of retransmissions over the network.

B. Threshold Based Iterative Pre-Copy

To improve the efficiency of lazy pre-copy, a set of criterion is used which decides whether or not to transfer memory pages from the source to the target host. Data transfer is predicated on threshold ***K***, which increases in value on a round by round basis.

While lazy pre-copy is executing the value of ***K*** is adjusted and set to one of three thresholds $K_1(\alpha\%)$, $K_2(\beta\%)$, and $K_3(\gamma\%)$. The maximum number of rounds for Xen is 30. These rounds are divided into three stages. The first stage (iterations $0\sim i_1$) uses threshold $K_1(\alpha\%)$, the second stage (iterations $i_{1+1}\sim i_2$) uses threshold $K_2(\beta\%)$ and the third stage (iterations $i_{2+1}\sim i_3$) uses threshold $K_3(\gamma\%)$. The thresholds for the three stages are allocated three different memory sizes. The first stage is allocated 5% of the memory. The second stage is allocated 1% of the memory. The amount of memory allocated in the third stage is defined by LPC which counts the number of dirty pages per round. According to which conditions are met and if they reach the associated threshold value, dirty memory pages are delivered to the target host.

The Multi-Phase Pre-Copy method described above is now presented in the following algorithm:

Algorithm 1: Multi-Phase Pre-Copy

```
Input:
Set L: the collection of page which is with 1
Set LP: the collection of page which is with 10
Set LPC: the collection of page which is with 100
Begin:
Pre-Migration and Reservation
Set $L$ = Set $LP$ = Set $LPC$ $\leftarrow$ NULL
```

```
For iteration 1:
    For page j:
        If ($P_j$ = 1) then set L ← $P_j$
For iteration 2:
    For page j:
        If ($P_j$ = 0 & $P_j$ in set L) then set LP ← $P_j$
    Reset set L
    For page j:
        If ($P_j$ = 1) then L ← $P_j$
For iteration 3 and afterward:
    Reset set_LPC
    For page j:
        If ($P_j$ = 0 & $P_j$ in set LP) then set LPC ← $P_j$
  Reset set LP
    For page j:
        If ($P_j$ = 0 & $P_j$ in set L) then set LP ← $P_j$
    Reset set L
    For page j:
        If ($P_j$ = 1) then L ← $P_j$
    If LPC > threshold then then send $P_j \in$ LPC to destination host //Pre-Copy
    If one of the termination conditions T1, T2 and T3 hold then stop_and_copy ()
```

5 Performance Analysis

In this section we present the experimental environment and experimental results of the proposed live migration mechanisms. Performance analysis was done by running the proposed algorithm on a simulated physical environment. To simulate the physical environment, multiple virtual machine images were installed and executed on a single physical machine.

A. Experimental Environment

To evaluate the performance of the proposed technique, we implemented the multiphase virtual machine migration algorithm and tested its performance on a physical machine running Xen version 3.1.2. All virtual machines had a single CPU core and tests were performed using different amounts of RAM. The specification of the physical machine used is shown in Table 1. The software environment is presented in Table 2.

Table 1. Physical Machine 1 Hardware Specifications

CPU	AMD Operton 6100 2.0G (8 Core) x1
RAM	8 GB
Disk	500 GB x2 (7200rpm)
Network	1 Gigabit

Table 2. Software Environment

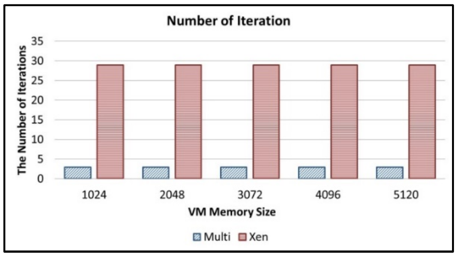
OS	CentOS 6.2 x86_64
Kernel	2.6.32
File system	EXT4

To evaluate the performance of multi-phase pre-copy, we implemented multiphase virtual machine migration and tested its performance against the pre-copy method implementation used by the Xen hypervisor.

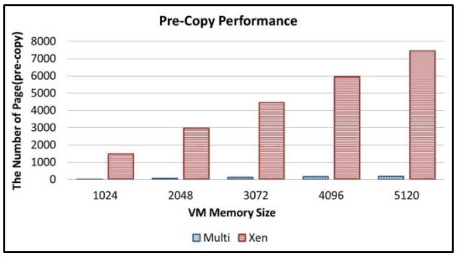
The pre-copy methods were tested to see how well they performed with different image sizes and with different dirty memory page rates. Metrics used to evaluate performance include: number of iterations, pre-copy performance (number of pages transferred), downtime and total migration time.

B. Experimental Results

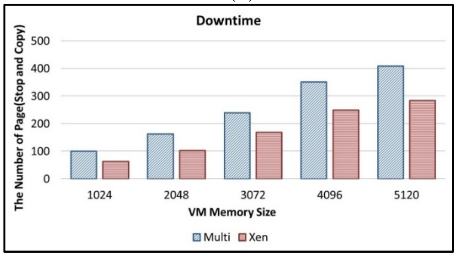
Real-time experimental results for LMVM were performed for Xen and multi-phase methods. Results show that when the the rate dirty pages occur is low, multi-phase pre-copy is able to reduce unnecessary data transfers significantly. When the dirty rate increases the number of dirty pages is higher, therefore the number of necessary data transfers increases. Consequently, the improvement in performance between Xen pre-copy and multi-phase pre-copy is reduced.



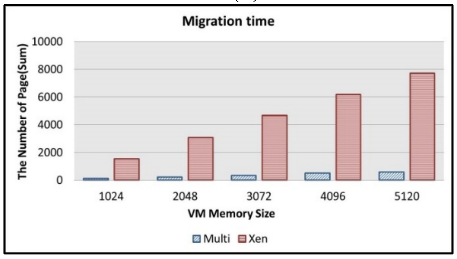
(a)



(b)



(c)



(d)

(a) Number of iteration (b) Pre-Copy performance (c) Downtime (d) Migration time

Fig. 4. Impact on Dirty Rate 0.05

Experiments were conducted to compare the performance between multi-phase migration and Xen's iterative pre-copy method. As shown in figure 6(a) regardless of virtual memory size, the number of iterations remains constant. However, In Figure

6(a) the number of iterations is much smaller for multi-phase pre-copy than that of Xen's pre-copy due to an earlier execution of a termination condition. Less iteration does not by itself equate to better performance, however it does mean less pages had to be transferred repeatedly over the network. Figure 6(b) refers to the pre-copy performance. Pre-copy performance refers to the number of data transfers over the network from source to the target host. Figure 6(c) show the downtime for multi-phase pre-copy is slightly longer than Xen's multi-phase pre-copy. However Figure 6(d) show that the total migration time is much less overall.

6 Conclusion

In this paper, we presented a multi-phase pre-copy method for live migration of virtual machines over a network. We show that compared to Xen's pre-copy method, multi-phase pre-copy is able to transfer dirty memory pages more efficiently from a source host to a target host. We show that multi-phase pre-copy can reduce the number of memory page transfers over the network with only a small increase in downtime. We also show that multi-phase pre-copy total migration time is significantly better than Xen's pre-copy method. For future work we intend to further improve live migration downtime and study how our algorithm performs on different network configurations.

References

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: SOSP 2003 Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, vol. 37(5), pp. 164–177 (December 2003)

[2] Xen-org, Xen, http://Xen.org/

[3] Akoush, S., Sohan, R., Rice, A., Moore, A.W., Hopper, A.: Predicting the Performance of Virtual Machine Migration. In: 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 37–46 (August 2010)

[4] Lin, H., Gao, W., Wu, S., Shi, X.-H., Wu, X.-X., Zhou, F.: Optimizing the Live Migration of Virtual Machine by CPU Scheduling. Journal of Network and Computer Applications 34(4), 1088–1096 (2011)

[5] Hines, M.R., Deshpande, U., Gopalan, K.: Post-copy Live Migration of Virtual Machines. ACM SIGOPS Operating Systems Review 43(3), 14–26 (2009)

[6] Hu, B., Lei, Z., Lei, Y., Xu, D., Li, J.: A Time-Series Based Precopy Approach for Live Migration of Virtual Machines. In: IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS), pp. 947–952 (December 2011)

[7] Harney, E., Goasguen, S., Martin, J., Murphy, M., Westall, M.: The Efficacy of Live Virtual Machine Migrations Over the Internet. In: VTDC 2007 Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, Article, No. 8 (November 2007)

[8] Goldberg, R.P.: Survey of Virtual Machine Research, vol. 7(9), pp. 34–45. IEEE Computer Society Press, Los Alamitos (1974)

[9] Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: Proceedings of the Linux Symposium, pp. 225–230 (2007)

[10] Clark, C., Fraser, K., Hand, S., Hansen, J.G.: Live Migration of Virtual Machine. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, pp. 273–286 (2005)

[11] Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S., Rosenblum, M.: Optimizing the migration of virtual computers. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), vol. 36(SI), pp. 337–390 (December 2002)

[12] Zayas, E.: Attacking the process migration bottleneck. In: Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, vol. 21(5), pp. 13–24 (November 1987)