KNOWLEDGECUDDLE PUBLICATION

International Journal of Research in Advance Engineering (IJRAE) – Volume-1 Issue -1

Process migration and load balancing.

Drashti Patel
drashtipatel6576@yahoo.in
Gujarat Technological University,
Gujarat, India.

ABSTRACT: Process migration is the act of transferring an active process between two machines Today in the real world while process is under execution it may happens so that the process gets stacked in between. If this happens for more than one process then there should be some mechanism that helps process to precede further .Here comes the IDEA of LOAD BALANCING using PROCESS MIGRATION and restoring the process from the point it left off on the selected destination node. Several implementations have been built for different operating systems. With increasing deployment of distributed systems in general, process migration is again receiving more attention in both research and product development. We use process migration to control over the load sharing, availability of long process, utilizing some special resources.

Keywords: Immigrant, Migrant, Refugee, Sparsely.

I. INTRODUCTION.

It may happens so that the process gets Today in the real while process is under execution stacked in between.

- If this happens for more than one process then there should be some mechanism that helps process to proceed further.
- Here comes the IDEA of LOAD BALANCING using PROCESS MIGRATION.

MIGRATION TERMINOLOGY: Process migration is the act of transferring process between two machines in which one is source and other is destination during its execution. Some architecture also defines a host or home node, which is the node where the process logically runs. Transferred state includes the process's address space, execution point, communication state and other operating system dependent state. Task migration represents transferring a task between two machines during execution of its threads. Remote invocation is the creation of a process on remote node. Remote invocation is usually a less "expensive" operation than process migration. Passive data represents traditional means of transferring data between computers; it has been employed ever since the first two computers were connected. Active data can be further classified into mobile code, process migration and mobile agents. These three classes represent incremental evolution of state transfer. Mobile code, such as Java applets, transfers only code between nodes.

II. ALGORITHM

A. EAGER COPY

The **eager (all)** strategy copies all of the address space at the migration time. Initial costs may be in the range of minutes. Checkpoint/restart implementations typically use this strategy, such as Condor or LSF. **eager (dirty)** strategy can be deployed if there is remote paging support. This is a variant of the eager(all) strategy that transfers only modified (dirty) pages. Unmodified pages are paged in on request from a backing store. Eager (dirty) significantly reduces the initial transfer costs when a process has a large address space. Systems supporting eager (dirty) strategy include MOSIX.

B. copy of reference

The COR strategy has the lowest initial costs, ranging from a few tens to a few hundred microseconds. However, it increases the run-time costs, and it also requires substantial changes to the underlying operating system and to the paging support. It is strategy is a network version of demand paging: pages are transferred only upon reference. While dirty pages are brought from the source node, clean pages can be brought either from the source node or from the backing store.

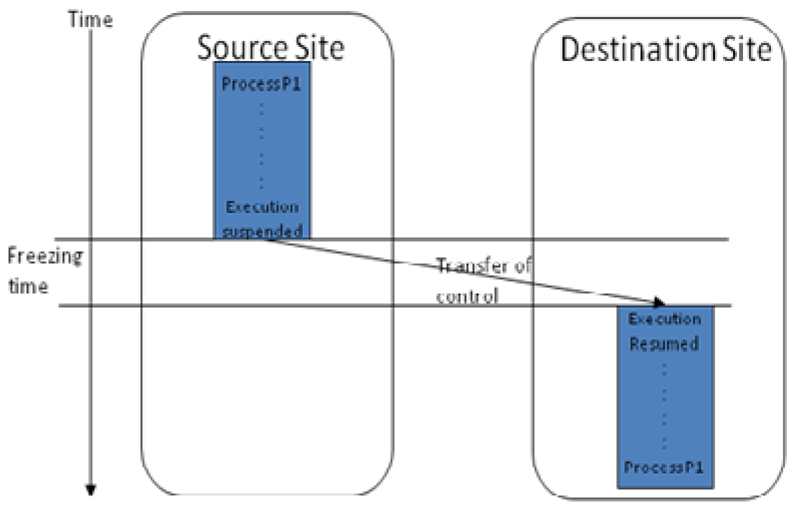
C. Flushing

The **flushing** strategy consists of flushing dirty pages to disk and then accessing them on demand from disk instead of from memory on the source node as in copy on- reference [Douglas and Ousterhout [3]]. The flushing strategy is like the eager (dirty) transfer strategy from the perspective of the source, and like copy on- reference from the target's viewpoint. It leaves dependencies on the server, but not on the source node.

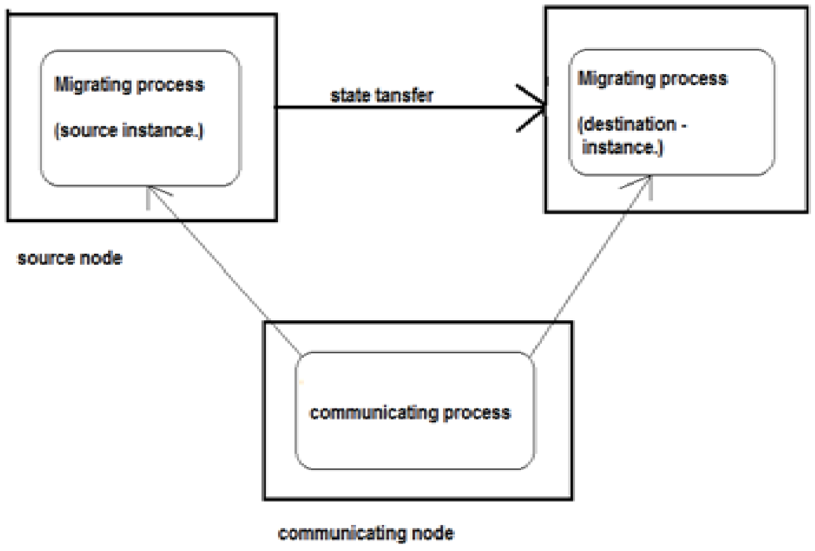
D. Pre Copy

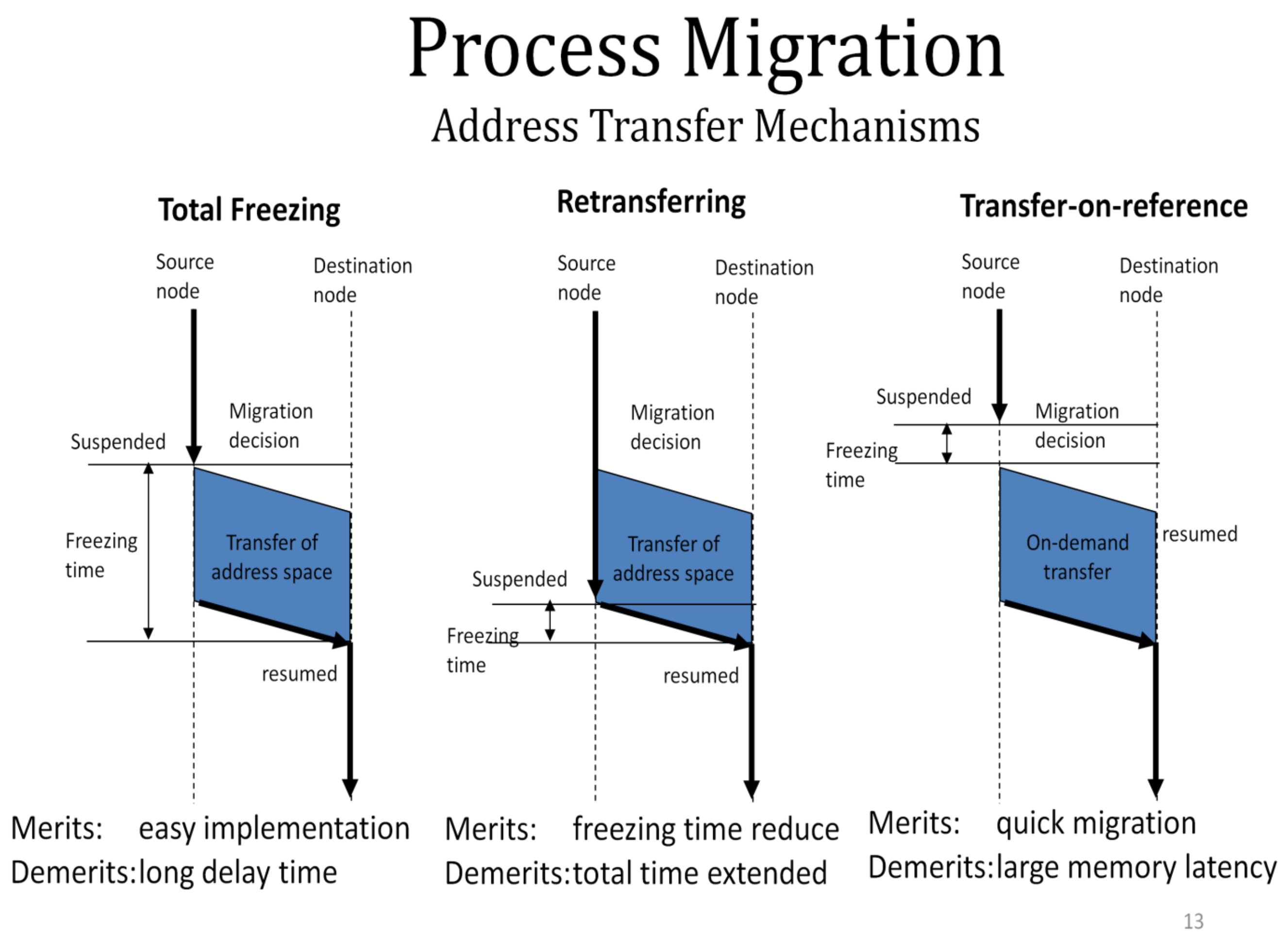
The **pre copy** strategy reduces the "freeze" time of the process, the time that process is neither executed on the source nor on the destination node. While the process is executed on the source node, the address space is being transferred to the remote node until the number of dirty pages is smaller than a fixed limit. Pages dirtied during pre copy have to be copied a second time.

FIGURE: 1 PROCCESS MIGRATION



- Selecting a process to be migrated
- Selecting the destination node
- Suspending the process
- Capturing the process state
- Sending the state to the destination
- Resuming the process
- Forwarding future messages to the destination


Migrating process
(source instance.)
state tansfer
Migrating process
(destination - instance.)
source node
communicating process
communicating node



III. Process migration enables

- D**ynamic load distribution**, by migrating processes
- From overloaded nodes to less loaded ones,
- F**ault resilience**, by migrating processes from nodes
- That may have experienced a partial failure,
- **Improved system administration**, by migrating
- Processes from the nodes that are about to be shut down or otherwise made unavailable, and
- **Data access locality**, by migrating processes closer to the source of some data.

IV. FUTURE

The goals of process migration are closely tied with the type of applications that use migration, as described in next section. The goals of process migration include:

Accessing more processing power is a goal of migration when it is used for load distribution. Migration is particularly important in the *receiver-initiated* distributed scheduling algorithms, where a lightly loaded node announces its availability and initiates process migration from an overloaded node.

Exploitation of resource locality is a goal of migration in cases when it is more efficient to access resources locally than remotely. Moving a process to another end of a communication channel transforms remote communication to local and thereby significantly improves performance. It is also possible that the resource is not remotely accessible.

Resource sharing is enabled by migration to a specific node with a special hardware device, large amounts of free memory, or some other unique resource. e.g. NOW for utilizing memory of remote node.

Fault resilience is improved by migration from a partially failed node, or in the case of long-running applications when failures of different kinds (network, devices) are probable [Chu et al., 1980]. In this context, migration can be used in combination with check pointing, such as in Condor.

System admonition is simplified if long-running computations can be temporarily transferred to other machines. For example, an application could migrate from a node that will be shutdown, and then migrate back after the node is brought back up. Another example is the repartitioning of large machines.

V. CONCLUSION

So, by transferring any process from one host or home node to another node we can use resources, memory and give CPU time equally. So we use process migration to control over the load sharing, availability of long process, utilizing some special resources.

REFERENCES

[1]. Dannenberg, R. B. and Hibbard, P. G. (July 1985). A Butler Process for Resource Sharing on a Spice Machine. *IEEE Transactions on Office Information Systems*, 3(3):234–252.

[2]. Hwang, K., Croft, W., Wahl, B., Briggs, F., Simons, W., and Coates, C. (April 1982). A UNIX-Based Local Computer Network with Load Balancing. *IEEE Computer.*

[3]. Klein rock, L. (1976). Queue Systems vol. 2: Computer Applications. *Willey, New York*.

[4]. Hildebrand, D. (April 1992). An Architectural Overview of QNX. *Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 113–126

[5]. OMG (March 1996). Common Object Request Broker Architecture and Specification. *Object Management Group Document Number 96.03.04*

[6]. Shapiro, M., Dickman, P., and Plainfossé, D. (August 1992). Robust, Distributed References and Acyclic Garbage Collection.

[7]. *Proceedings of the Symposium on Principles of Distributed Computing,* pages 135-146.

[8]. Shapiro, M., Gautron, P., and Mosseri, L. (July 1989). Persistence and Migration for C++ Objects. *Proceedings of the ECOOP 1989–European Conference on Object-Oriented Programming*.

[9]. Shivaratri, N. G. and Krueger, P. (May-June 1990). Two Adaptive Location Policies for Global Scheduling Algorithms. *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 502–509