Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 9; April-June, 2015 pp. 25-28

http://www.krishisanskriti.org/acsit.html

Process Migration in Cloud Computing

Pankajdeep Kaur[1] and Anita Rani[2]

[1]CSE Department GNDU, RC, JALANDHAR
[2]M.TECH CSE(2nd SEM) GNDU, RC, JALANDHAR
E-mail: [1]pankajdeepkaur@gmail.com, [2]anitasaroay@yahoo.in

Abstract—*Cloud computing is the delivery of computing services over the internet. Cloud services allow individuals and other businesses organization to use data that are managed by third parties or another person at remote locations. In cloud computing, Process migration is a technique in which an active process is moved from one machine to another of possibly different architecture. Process migration is used to load sharing among the pool of processors and also improve the communication performance, reduced the cost by bringing the processes closer together. Process migration needs to capture the process's current state of execution and recovering it on the destination machine in a manner understandable to it. This paper includes the mechanism of process state capture and state recovery.*

Keywords*: Process Migration, Initiation, State Capturing, Recovery.*

1. INTRODUCTION

Process Migration is a technique in which an active process is moved from one machine to another of possibly different architecture. Therefore it is necessary to capture the process's current state of execution and recovering it on the destination machine in a manner understandable to it. Process migration is used to load sharing among the pool of processors and also improve the communication performance, reduced the cost by bringing the processes closer together. This paper describes process capture and recovery of the internal state of process.

Further this paper highlights major migration issues include:

1) The mechanism of process state capture and recovery.

2) The initiation of the process state capture.

Process state capture in distributed computing systems cannot simplify the process state capture, when the request of capture is received. The process state can be capture and initiated only at certain points –

1) When different instances of computation of different architecture's points are equal – so that the process can be again started from the same point where it is halt.

2) When the request for capture is received, then the state capture is initiated at the next point. At the same time, the normal execution of process should be ensured.

A novel approach is used to process state capture and recovery, in which the state capture and recovery is maintained. When high performance computing applications are used, the performance is achieved by the elimination of polling within critical loops and it should be significant. This solution is also capable for effectively enabling all points of equivalence that are present in a computation if there is minimal latency. In a polling approach, to get this minimal latency, at all potential points poll- points are to be placed, in which, the performance overhead is incurred during normal execution of the process. One of the main features of this approach is automatic transformation of the process's program code to state capture and recovery functionality that is uncooperative. This modification is performed at the platform-independent intermediate level of code representation but it preserves the original program code. The main properties of this approach are portability, ease of use, flexibility and application-specific requirements.

A mechanism that solving the process state capture and recovery problem will generate a checkpoint for an active process. This contains a complete description of that process's state and point in execution. It also supports later use of that checkpoint to restart a process, possibly on different type of computer from which the original checkpoint was created. [1]

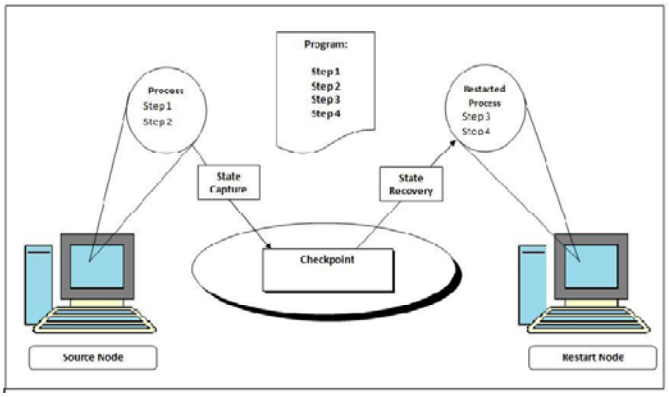

Fig. 1: Basic Operation of State Capture[1]

Fig1 depicts the basic operation of a heterogeneous state capture and recovery mechanism. The basic operation of state capture is to recover the state of process at destination using the checkpoint .Checkpoint is depicts the basic operation of a

heterogeneous state capture and recovery mechanism contained the information of various processes and also helps to recover the states of process at restart nodes when the process migration is going to be implement. But it is not easy to capture the initial state of the process. It can be done different instances of computation of different architecture's points are equal.

2. RELATED WORK

For homogeneous computing systems, Process state capture and recovery mechanisms are well developed and can now typically be performed with minimal overhead and latency; less progress has been done in this functionality across heterogeneous architectures.

The Tui System[2] constructed to provide a heterogeneous migration tool that use on four common architectures in the UNIX environment. In this system, the capture and recovery of the state of a process is done in a highly machine dependent manner, that requires complete knowledge of the fixed conventions. Also, when a process state capture is requested, a breakpoint instruction to trap to the capture mechanism is placed at all "pre-emption points" i.e. the enabled points of equivalence. These are available in the program to effect capture initiation. In such approaches, on architectures with varying instruction lengths, it is possible that the breakpoint instruction placed at one pre-emption point overwrites the instruction placed at another point or the current instruction. To avoid this loss of correctness, it is necessary to fill space at each pre-emption point and it should be enough to adjust the breakpoint instruction. Therefore it is necessary to placing dummy instructions, which introduce performance overhead during normal execution of the program.

The process introspection model proposed by Ferrari the portable[5] (or shadow-) check pointing model presented in perform the activation history state's capture in an machine-independent manner by using the call-return functions provided in the high-level programming language and it can also done by the intermediate instruction set. Here, the architectural differences are occurred it will be handling by the compiler. However, they used a polling approach to process state capture initiation. Poll points are created there, where the process determines if a capture can be initiated or not. In such an approach, a valuable amount of performance overhead would be incurred during normal execution because the continuous polling system is used.

In applications load balancing policies and logging mechanism for fault tolerant systems, long latencies are not acceptable. In the case of load balancing, e.g., the main purpose of migrate the process one server to another server so that load will be reduced on the system as soon as possible. For such applications with a minimal latency requirement, almost all potential points of equivalence present in the computation must be effectively enabled. The polling approach would not be suitable because the performance overhead due to persistence polling at all such points would reach severely unacceptable levels. There are several unacceptable levels. Therefore, it is necessary to design new approaches.

A new poll free solution is given by P. Bungale[4] in which they propose a poll-free solution that involves preserve semantics self-modification of code. However, unlike the other poll-free solutions, when a capture is requested it do not place capture initiation instructions at all enabled points of equivalence. Instead, it places the initiation instructions in a copy of only a small portion of the code. Thus, it removes the introduction of placeholder dummy instructions and thus it increases the performance overhead.

The selection of the enabled subset from the entire set of potential points of equivalence is to be done according to the requirements of application and constraints. For example,

1) An application which desiring minimal latency can effectively enable all potential points of equivalence that are present in a computation and this work could not be done by a polling approach, because of the high performance overhead during normal execution.

2) In the case of high-performance computation applications, only a subset of the potential points of equivalence would have to be enabled in order to utilize the performance enhancements achieved and especially within critical loops and through machine-dependent optimizations across the potential points of equivalence but they are non enable.

One of the major advantages of this design is the modification of program to incorporate state capture and recovery function. It giving processes the ability to save and restore their states automatically after they initiated. Some assumptions are also taken in this approach,

1) The process is based on a program which can be written or translated to an imperative, stack-based intermediate representation to which transformations will be done by a compiler and also possibly used a programmer.
2) The main element of this design is a table that mapped all the enabled points of equivalence to the corresponding points of object code. [4]

A new poll free solution is proposed by D. Ajmire [6], in this by using the various assumptions are made and algorithms are designed by using which process state initiation, capturing and recovery is done.

Assumptions that are used in this design-

1. Compiler helps in obtaining the equivalence point table.

2. Operating system supports in generates the current value of program counter for a process's state initiation.

3. Machine dependent optimizations across enabled points of equivalence shall not be allowed because they prevent the different versions of the program's compiled across heterogeneous platforms.

a) State Capture Initiation Algorithm-When a request for a process's state capture is received from an external source, the following steps are taken:

1. The current value of the process's program counter is obtained using operating system helps.

2. The program counter value is used to identify the current executing function and the current segment of code between two points of equivalence containing the current instruction.

3. The following steps are carried out for state capture initiation on next point of equivalence:

b) State Capture Algorithm-Once the control transferred to the state capture function, the following tasks are performed initially:

1. Save the initiated point of equivalence and the interrupted function of capture.

2. The new address of the current activation record is replaced by the saving back copy of the interrupted function.

c) State Recovery Algorithm-Once a new process has been created on the destination machine, the following steps are taken into consideration to perform the process state recovery:

1. A jump instruction with destination as the corresponding restoring back copy is placed at the entry point of each function

2. When first activation record is created, the epilogue will restore the static data from the checkpoint file.

3. The restoring epilogue performs the tasks.

4. Step 3 is repeated until all activations history have not been restored.

5. The process finally resumes normal execution.

Once all the activations have been restored, the original function entry points are restored and control is transferred to the point of equivalence at which the state capture had been initiated.

D.S. Miloji [7], proposed a Migration Algorithm to migrate the process from one machine to another without state capture but it uses forwarding references.

1. **A migration request is send to a remote node** after negotiation, migration request has been accepted.

2. **A process is apart from its main node** by suspending its current execution and temporarily redirecting communication as described above.

3. **Communication is temporarily redirected** .This step continues with steps 4, 5, and 6, as long as messages received.

4. **The process state is extracted,** by memory contents processor state, communication state. The communication state is system dependent. Local system's internal state is not transferable. The state of process is retained on the source node until the migration is not complete but in some systems it remains, after migration completes.

5. **A destination process instance is created** in which the new state will be imported. Until a sufficient amount of state has not been transferred from the source instance, destination instance is not used. After sufficient amount of state has been transferred, the destination instance will be promoted into a regular process.

6. **State is transferred to a new instance,** all the states are not transferred but some of the state can be brought even after migration completion.

7. **Means of forward references** to the migrated process should be maintained**.** It is also necessary to communicate with the process and to control it. This step also uses communication channels at the destination and redirected after step 3.

8. **New instance is resumed only** when sufficient state has been transferred. After this step, process migration completes. When all of process state is transferred from the original node, it can be deleted on the source node.

By N. Tzirita [8], a new approach is proposed in which resource consumption can be minimizing by using process migration.

Minimizing resource consumption using process migration- pay-per-use model adopted in clouds, according to this, how much resources an application running is used in cloud computing environment, the greater the amount of money will be charged to the owner of application. Therefore, intelligent solutions must be used to minimize the resource consumption. The problem which is common, to identifying an assignment scheme between the interacting components and the computing nodes of a cloud environment, to minimize total amount of resources consumed by the application. Various centralized solutions are found unsuitable for large-scale applications. Therefore, a distributed algorithm is important to assignment resources for minimum resource consumption.

For minimizing resource consumption following methods are used to solve:

(a) Centralized Solutions.

(b) Distributed Solutions.

(c) Migration Mechanisms.

a) Centralized Solutions-It includes:

i) Task allocation- Older works focused on solving the task assignment problem over a dual processor system such as to minimize the total communication cost.

ii) VM placement- There is also a significant number of works in the literature related to energy consumption and VM

consolidation. However, most of them do not take into account the network overhead when deciding about the assignment of VMs onto nodes (servers).

b) **Distributed Solutions**-Over the last few years, large-scale distributed systems (e.g., clouds, grids, sensor networks, etc.) have gained a lot of attention. Centralized solutions were rendered useless due to scalability issues, and were finally superseded by distributed ones.

c) **Migration Mechanisms**-In the following text, we mention the most relevant works we found in the literature in regards to the migration mechanisms that they are supported at different levels:

i)Thread level.

ii)Process level.

iii)VM level.

Advantages-The advantages of using process migration are-

1. Efficiency-To providing efficient state capture and recovery, the run-time performance overhead introduced by the mechanism should be at acceptable levels. If checkpoints are not performed in execution, then it will not run significantly and slower without this service.

2. Generality- The mechanism is appropriate for a wide range of architectures and a wide variety of programs ,written in a multiple languages

3. Suitability for Minimal Latency-The state capture mechanism should be suitable even for ensuring minimum possible latency that is the time delay between when a capture initiation is requested and when the capture is actually initiated. This is the time taken to reach at the next possible point of equivalence in the computation.

4. Ease of Use- This mechanism is fully automatic, requiring little effort on the part of the application programmer. Such full automation is only possible for programs that expressed in a platform-independent manner. [6]

3. FUTURE SCOPE

The presented work is done in process migration perform the scheduling and the allocation of the processes to the clouds. Both in case of under load and overload conditions. But, in case of over load condition, the process migration is performed from one cloud to other.

The Future scope of the work is possible in the following directions -

1) The presented work is defined the overload conditions in terms of deadline as well as the memory limit of the clouds. In future some other parameters can also be taken to decide the migration condition.

2) The presented work is defined for the public cloud environment, but in future, the work can be extended to private and the hybrid cloud environment.

4. CONCLUSION

This paper describes an approach to process state capture and recovery in heterogeneous computing systems that achieve minimum performance overhead during normal execution of the process. This presents being poll-free, is suitable even for an application desiring minimal latency as it can afford to effectively enable all points of equivalence presents. For minimizing the resource consumption the solutions are used to solve the problem. These approaches thus achieve the minimum performance overhead during execution of process. The presented work is about to perform the scheduling and the allocation of the processes to the clouds in case of under load and overload conditions. In case of over load condition, the migration of the processes is performed from one cloud to other.

REFERENCES

[1] Ms. Alankrita Aggarwal,"Comparative Review of Scheduling and Migration Approaches in Cloud Computing Environment", 2014 IJEBEA International Association of Scientific Innovation and Research (IASIR)

[2] T.Velte and A.Velte and R.Elsenpeter, "Cloud Computing, A Practical Approach, New York", USA, McGraw-Hill, 2010.

[3] K. Sammy, R. Shengbing and Ch. Wilson, "Energy Efficient Security Preserving Process Migration In Cloud Computing", International Journal of Computer Science Issues (IJCSI), March 2012

[4] Prashanth P. Bungale, Swaroop Shridhar. "An Approach to Heterogeneous Process State Capture and Recovery to Achive Minimum Performance Overhead during Normal Execution." Proceeding of the International Parallel and Distributed Processing Symposium(IPDPS'03)

[5] A.J. Ferrari, Stephen J. Chapin, and Andrew S.Grimshaw, "Process Introspection: A Heterogeneous Checkpoint / Restart Mechanism," Department of Computer Science, University of Virginia.

[6] D.Ajmire, M. Atique"Task migration in Cloud Computing", International Journal of Innovative Research and Development,2013.

[7] D.S. Miloji, F.Douglis, Y. Paindaveine, TOG Research Institute, EMC, and "University of Toronto and Platform Computing".

[8] N. Tzirita, S.U. Khana, Thanasis Loukopoulos "On minimizing resource consumption of cloud applications using process migration" Journal of Parallel and Distributed Computing,2014.

[9] V. H Pardeshia,"Cloud Computing for Higher Education Institutes: Architecture, Strategy and Recommendations for Effective Adaptation", Symbiosis Institute of Management Studies Annual Research Conference,2014.

[10] L.Gkatzikis," Efficient Task Migration Policies for Cloud Computing Systems", Education and Lifelong Learning, 2014.