UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

jMAD- A small Java Mobile Agent Development platform

[Abhishek Yenpure,Dhiraj Patil, Omkar Jadhav, Ajinkya Bhat, Nitin Umesh]

***Abstract--Grid Computing has been around for as* long as the modern computers themselves yet have failed in obtaining a clear standard which would define its implementation across the computing domain. Our project focuses on the Process Migration aspects of Grid Computing/Clusters for Java Applications. The available solutions for Java Process Migration require high level of Java Expertise for the user itself let alone the developer. These solutions also require changes to be made in the JVM (Java Virtual Machine) which is a hassle for SMEs who cannot afford to put such high level solutions into work. Our Project Solves the problem by establishing a Method of Process Migration which will not need any changes in the JVM and will not require the user to know the internals of the implemented method. The user will only have to worry about developing his code remaining totally aloof of what will be done to it later.**

Keywords--JPM,Object Serialization, CodeInstrumentation,Migration,Broadcast, JVM (Java Virtual Machine)

I. *Introduction*

A. *Problem*

Process Migration is the prime need of fault tolerant systems running in Grid Environments and Clusters. The standardization of such concepts and methods used to achieve the goals of such concepts are very difficult because of the vividness of the implementations and different perspectives of the ideas used to implement the same thing in different environments in terms of technologies (Languages, Architectures, etc.). In environment running/developed on languages like C/C++/Fortran the process Migration is much easier than the new higher level languages like Java etc.

In Java the main problem is that it does not let you play around with the process its properties like address spaces etc. for security purposes and hence said to be running in a sandbox [3]. Unlike C/C++/Fortran Java does not allow access to method stacks and Program counters to track the program. This leads to a problem of Process Capture which will help in saving the executing process and resurrect it according to the need of the user.

AbhishekYenpure,DhirajPatil,OmkarJadhav,Ajinkya Bhat,Nitin Umesh
Sinhgad Institute of Technology and Science,University of Pune
India
{abhishek.yenpure@hotmail.com , dhirajpatil.18@gmail.com , omkar7272@gmail.com , ajinkya@blog404.com , nitinumesh8391@gmail.com}

Hence because of the above problems we needed to address the Process migration problem in Java. It’s not that such a thing was never tried before; the difference is that all the existing implementations require the implementers to significantly change the JVM (Java Virtual Machine) itsimplementation [1]. This leads to another headache of how the novice users/developers of the system will deal with a thing highly technical like specifications of JVM/Java Language with bare minimum resources as in case of a lot of SMEs.

The Problem is how to migrate the Java Processes without modifying the JVM in any way and without the user needing to deal with the technicalities of the implementation so that he will have to worry only about his own Program its implementation and not how the system will handle his program.

In this project we are going to handle the exact need of the day and will be trying to implement a platform for development of Mobile Agents to achieve Process migration in Java using Method Level Granularity.

B. *Solution*

Java does not allow us to access the address spaces of the current objects in execution discouraging the system help for capturing the objects in execution. But however java provides various inbuilt interfaces which allow us to store the state of the current executing objects in Files which may be transferred across systems and can be used to resurrect the objects and form a part of some other process, this technique is called as ‘object serialization’ and the interface provided by Java for this is ‘java.io.externalizable’.

Having thus solved the problem of savingthe executing state, tracking of the process can be done using checkpoints. We can allow the whole process to run between checkpoint to checkpoint forming executable granular units in the same method. This will help in resuming the process on other machine by starting its execution form the checkpoint it had ended on previously. Checkpoints can be implemented in the .java file or the .class file of the user the parts of which are discussed further in section *II.A and II.C*.

The whole approach provides seamless migration of Java process running on a cluster only introducing the overhead at the time of the class load event or the compilation of the class depending on the approach of the introducing code in the user files for process capture and tracking (introducing checkpoints).

UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

Various performance tests have been done on the already similar system [1] which our project is based on and the results were obtained positive thus supporting our take on the system which would yield in a high performance implementation of the Grids and Clusters.

C. *Scope and Visibility*

The project can take a really big scope if we take into consideration all the parameters of the Grid Process Implementation and Cluster Application deployment (as you will find in MOSIX etc.) into account and hence we will be limiting ourselves to a few select and important aspects of the implementation as we lack much insight, the major part of which will be dealing with the Java processes only.

In terms of the visibility of the project we will be able to deliver whatever we will be promising in the solution to the problems identified, doing justice to the project topic as well as the project implementation and its business value.

We will hence be spending time to make our project as transparent, robust as possible, so for it to become a part of the new ecosystem of middleware and system software which will be completely purpose built.

As a part of the commercial feasibility of the project we will be conducting various performance tests on our implementation and will be comparing our approach of implementation with "M-JavaMPI" [1] on which our project is based on. The project is to introduce us to new technologies used in industries for application development like bytecode instrumentation, reflection and introspection, object serialization apart from all the trivial technical aspects of the project and will give us an insight of industrial application development and understanding computing in a whole new way.

II. *Implementation*

The way we are going about in the system is by appointing java code instrumentation before compilation(not to be confused with Compile time Bytecode instrumentation) to insert checkpoints to track the progress of the program and object serialization [2,4,6] for the storage of objects and other important runtime information, by appointing Java only features. The developers can gain a great deal in debugging the system as well as the system will provide the user with the instrumented code too, unlike the JVMDI approach used in M-JavaMPI [1].

A. *Pre Processor*

The pre-processor will use the support of Java Reflection [2,5,6] to get all the Fields and Methods declared inside the Program,thus it will help cache all the variables,objects inside a class, while the user has no need to write his class to be externalizable,the pre-processor will be responsible to do that by instrumenting the java code, the java code will now necessarily need to implement the methods declared in the interface, hencewe will need to write the two methods readExternal() and writeExternal() [4,6] explicitly. These methods will help in the storage of all

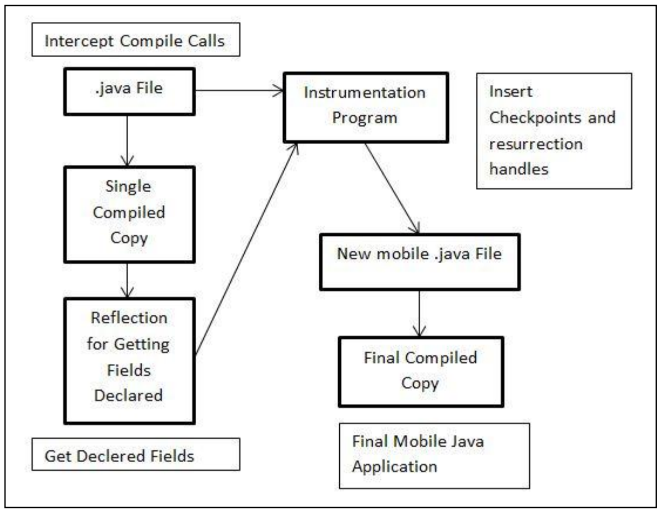

Figure 2.1. Schematic of the Pre Processor

the objects in the current Java Process. To write all these objects to the storage, here we will need to know all those objects by the method Class.getDeclaredFields()[2,5,6] which returns all the Fields declared within the class, hence after we know the names and types of all the objects in the class, we would write them to the storage by virtue of writeExternal().This will be done only when the System is overloaded. These files after storing to the file system will then be sent to the node which is willing to accept the process form the current Node and then the accepting node will resurrect all the objects by using the method readExternal(). The Pre Processor may make the code look really verbose, but it might not at all put a lot of efforts on the system then it did earlier.

Apart from just declaring these two methods, thePre-processor will also be responsible for the insertion of checkpoints in the program.

B. *Object Serialization*

Object Serialization [2,4,6] is the Method provided by java for the storage of objects or synchronization of objects in the current execution context of the Java Virtual Machine the technique uses two of java's interfaces

- java.io.serializable
- java.io.externalizable

In the above two interfaces externalizable implements serializable interface too so that whenever the externalization of object is to be performed, the objects are serialized and would not require any other mechanism for locking the objects under externalization. The externalizable interface provides two methods for the storing and resurrection of the objects of the classes that implement the interface. This is necessary that whatever objects are to be saved need their classes to implement the externalizable interface of java. This technique will help us to store the state of the objects currently in the execution.

Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

C. *Checkpointing*

The approach for keeping the track of the process during the execution M-java MPI [1] appoints are the services provided by the JVMDI (now JVMTI) which gives access to all the regular program execution parameters (Stack, Program Counter), but in our approach, we are implementing this using static variables as checkpoint storage and then using named blocks/conditional loops for forming one logical block of execution,. This will enable the original author of the program to gain transparency into the codes instrumentation and hence will put him in control again.

III. *Architecture*

A. *Architecture of the System*

Based on the system's requirement and referring to the paper "M-JavaMPI" [1] the architecture we see of our system is somewhat like the Fig. 3.1 The Systems has the Following basic and predominantly important blocks:

- Pre-Processing Layer
- Migration Layer
- Communicating Layer

The functions of which will be briefed eventually in the further briefing of the architecture and mechanisms to be adapted by us in the system implementation.

B. *Communication between Nodes*

Fig.. 3.2 shows the communication will take place in the system and what messages will be sent by the nodes. Initially, each machine communicates all its processor and memory information/status to all other machines using a UDP message. Consider machine A from Fig. 3.2, it broadcasts a UDP message containing the information about its processor and memory information/status. In response, the other machines send a UDP message to machine A and

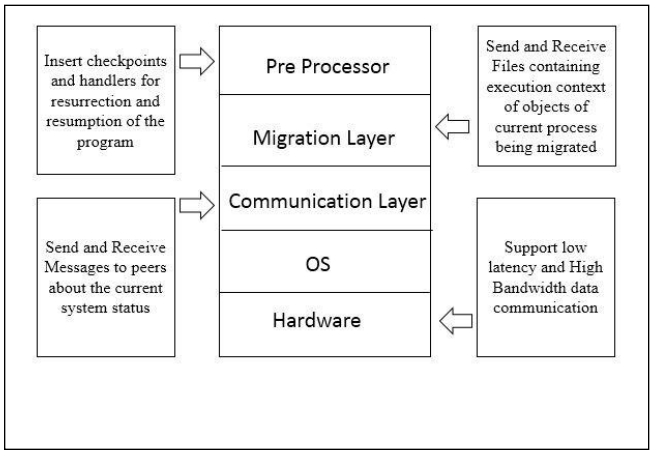
Insert checkpoints and handlers for resurrection and resumption of the program
Pre Processor
Send and Receive Files containing execution context of objects of current process being migrated
Migration Layer
Communication Layer
Send and Receive Messages to peers about the current system status
OS
Support low latency and High Bandwidth data communication
Hardware

Figure 3.1 Architecture of the System

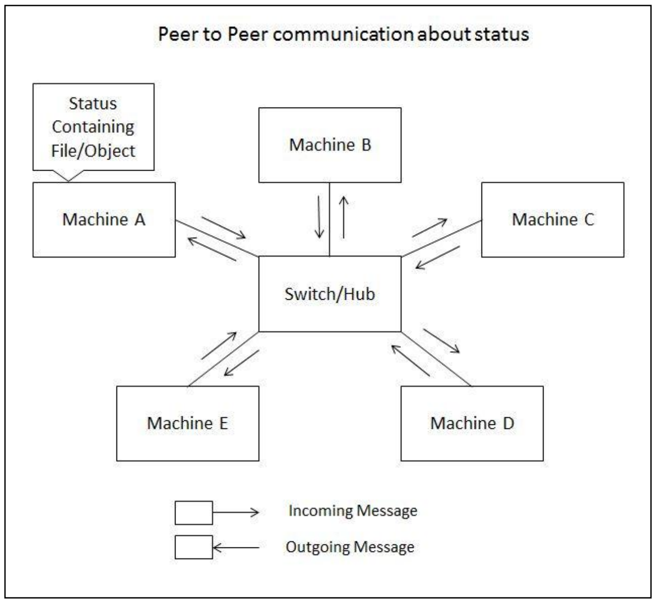

Figure 3.2. Communication between Nodes/Prres

all other Nodes, containing the status of their processor and memory usage. Machine A stores the status of these machines in an Index file. From this Index file (in the actual implementation, we will be using IP-Value pair mapped in a HashMap in Java Collection interface), it selects the machine for process migration having the least processor and memory usage. The information about the Processor and memory utilization is updated periodically (after every 10 sec say) so that the system can be updated regularly without over populating the channel and letting resources for communication by waiting too short but also not letting the systems change their resource utilization by waiting too long.

This trade-off between communication times is very essential because the system cannot be risked for just communicating and then letting the process to a higher powerful machine sacrificing its own resources in communication itself.

C. *Migration Initiation*

Fig. 3.3 describes the initiation of process migration. Now depending on the entries made in the Index file of machine A, the machine having the lowest value in terms of its status (processor and memory usage) is chosen. After getting the machine to migrate, the object saving process begins with object serialization. As described earlier the object Serialization is responsible for the saving of the objects of all the classes on the current running process. It's necessary to track all the possible classes the process is going to refer to before actually running the process so that proper provisions could be made so that the objects of those classes are saved and sent with the process to the other end and will not throw any run time exceptions on the other machine.

UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

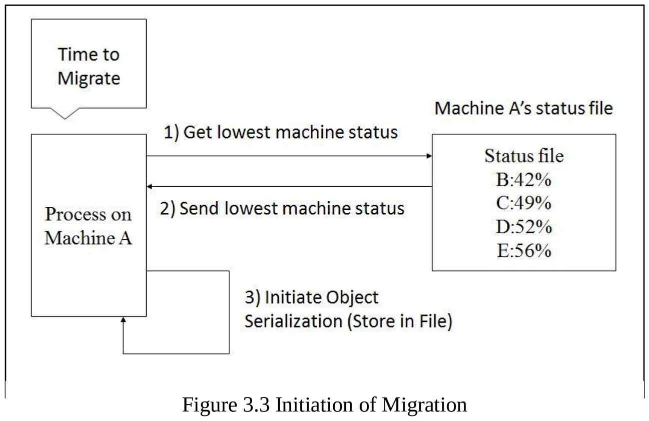
Time to Migrate
Machine A's status file
1) Get lowest machine status
Status file
B:42%
C:49%
D:52%
E:56%
Process on Machine A
2) Send lowest machine status
3) Initiate Object Serialization (Store in File)

Figure 3.3 Initiation of Migration

This we intend to do with the concepts of Reflection and Introspection in java. The addition of the pre-processor layer will ease the things by changing the program of the user sufficiently so that the problems of referencing are solved by us completely. The Fig. 3.4 shows the schematic of what was said previously about the referencing problem of the objects of other classes in the process.This approach is convenient in one way by establishing the channel between the main() methods of the two machines for the resurrection process as the objects in the main can be stored and forwarded and by using special communication protocols the main() methods can communicate with each other and then form the objects explicitly without resurrecting them but by querying the objects of the other main() method however this may have various network and security issues to be dealt with, incurring extra cost on the system.

This is also needed to be known that we are dealing with granularity as a certain block of statements and not at a statement level unlike M-JavaMPI [1] because without the support of JVMDI (now JVMTI) it is not possible to achieve that, and to a certain extent, it keeps the development of our proposed system a bit low in terms of complexity. This also frees us from the worry of capturing the

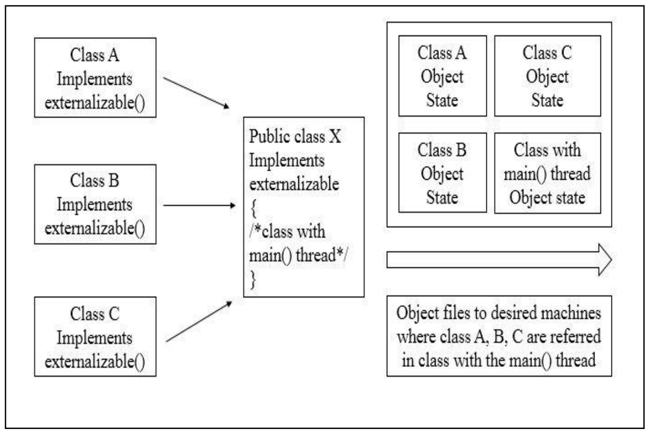
Class A Implements externalizable()
Class B Implements externalizable()
Class C Implements externalizable()
Public class X Implements externalizable { /*class with main() thread*/ }
Class A Object State
Class C Object State
Class B Object State
Class with main() thread Object state
Object files to desired machines where class A, B, C are referred in class with the main() thread

Figure 3.4 References in class with the main() Thread

intermediate results of the statement execution and storing them too while we try to migrate, hence instead letting the group of statements form that set run on the system and do all the changes to all the objects that occur in that block and then store them and migrate which reduces the system overhead for doing unnecessary thing than executing what is important in our perspective. This ultimately benefits the node over which the execution is being shifted by reducing the no of statements that it requires to execute. Also bytecode instrumentation if not done properly give rise to a whole new class of exotic errors and exceptions, making our approach a bit more secure too.

D. *Migration Exchange*

Fig. 3.5 objects that are to be saved by serialization are saved in the files by using the writeExternal() [2,4,6] method specified in the java.io.externalizable interface. The argument to the method is the ObjectOutput which is an interface implemented by the ObjectOutputSteram class so that the Objects can be stored in the files. After all those prerequisites of saving the object files, the files are sent to the other machine for resurrection, which is done by the method of readExternal() [2,4,6] method specified in the java.io.externalizable interface. After the resurrection of all the objects, the process could finally start the execution once again. Fig... 3.5 can be described as; machine A sends the object state file to the (chosen) machine B. Machine B resurrects the objects and then begins the execution of the process from the last saved checkpoint. After completion of the process, the updated object state file is sent back to the parent machine A.

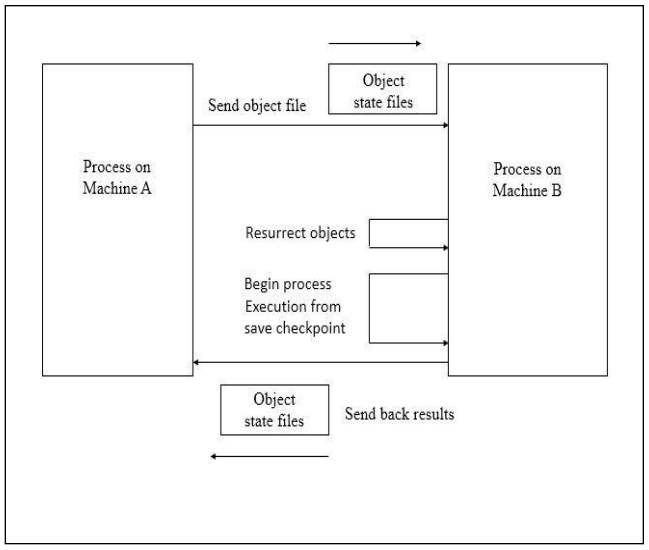

Figure 3.5Steps involved in migration of Process

IV. Conclusion

The Technology we are trying to develop might lead to significant changes in the paradigm of mobile agent development where the applications will be developed without bothering about the Process Migration but will be instrumented for additional functionalityand willyield ingreat load distribution/balancing by making the application into a Mobile Agent.

V. Future

After sufficient tests for the initial deployment we plan to make a library out of our project just like some well-established libraries such as JADE, and we are also planning to carry this project further as an Android Hypervisor for running huge tasks on Handheld devices.

References

[1] Ricky K. K. Ma, Cho-Li Wang, and Francis C.M. Lau. “M-JavaMPI: A Java-MPI Binding with Process Migration Support” presented at Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium

[2] Herbert Schildt “Java 2:The Complete Reference” , Tata McGraw Hill,2002

[3] Oracle Technology Network,” Java Language and Virtual Machine Specifications”, Internet: “http://docs.oracle.com/javase/specs/”.

[4] http://www.jusfortechies.com, Object Serialization,Internet: http://www.jusfortechies.com/java/core-java/externalization.php, [Sep 18,2012]

[5] Reflection in Java, Internet : “radio.javaranch.com/val/2004/05/18/1084891793000.html”

[6] JAVA 2 SE 7 Documentation(Oracle Java Documentation), Internet : “http://docs.oracle.com/javase/7/docs/api//”

About Author (s):

AbhishekYenpur
e

“jMAD –A small java Mobile Agent Development platform” is being researched and developed by us on part of our graduation project, and I am honored to bring our research to all the readers.

M NitinUmesh

Working on lower layer modules, close to the system was very interesting and challenging. I am very privileged to be a part of the jMAD team. I hope it contributes to the Grid Computing and Mobile Agents paradigm.

AjinkyaBhat

Working on the Cluster Computing phenomena has been a learning experience. Though we faced many ups and downs, we always came up with amazing solutions.

DhirajPatil

Being the part of jMAD team, I am highly obliged to present our paper in this prestigious platform. I personally enjoyed working on the communication module of our project.

OmkarJadhav

It has been an exciting experience working on the Migration module of this project and a sheer pleasure being a part of this jMAD team.