

Java Process Migration (JPM)

Mobile Computing Energy Improvement through JPM

Abstract

This document describes the system design for the Java Process Migration solution presented by Willard Thompson and Charles Weddle for COP5611 Advanced Operating Systems at Florida State University. The focus of the design is to improve mobile computing energy efficiency by introducing optimal process scheduler algorithms, process transmission logging, and process caching.

Java Process Migration (JPM).....	1
1. Introduction.....	1
1.1. Description.....	1
1.2. Section Overview.....	1
1.3. Terms, Acronyms, and Definitions.....	1
2. Java-PM Process.....	3
2.1. Description.....	3
2.2. Logical Design.....	3
2.3. Physical Design.....	5
2.4. Java-PM Process Sequence Diagrams.....	5
3. Java-PM Daemon.....	8
3.1. Description.....	8
3.2. Logical Design.....	8
3.3. Physical Design.....	10
3.4. Sequence Diagrams.....	11

1. Introduction

1.1. Description

The Java Process Migration (JPM) project is an effort to improve energy usage in a mobile computing environment through optimized process migration. Ultimately, by efficiently running a processor that meets the demand and does not run idly, the amount of energy consumed can be reduced.

The JPM project has identified three areas of improvement that will hopefully optimize process migration in a mobile computing environment so that energy can be conserved. First, through an optimal process scheduler algorithm that takes into account process size and processor cycles to completion, processes can be scheduled more efficiently that takes into account energy consumption.

Second, by grouping processes together in a process block for transmission can reduce the overhead spent transmitting processes to another computer for execution. This is called logging the processes for transmission.

Lastly, by caching processes on the remote computer, the process to be migrated will not even have to be transmitted. Process caching can have a large impact on energy conservation.

1.2. Section Overview

There are two main sections to this document, Section 2 that describes the Java-PM process and Section 3 that describes the Java-PM daemon. Section 2 goes into detail about the Java-PM process logical and physical design. The abstract class used to implement the Java-PM process framework is also described in detail in this section.

Section 3 describes the logical and physical design of the Java-PM daemon. The different functional components that make up the Java-PM daemon are described in detail. The interfaces used to for inter component communication within the Java-PM daemon are described as well.

1.3. Terms, Acronyms, and Definitions

The following are terms, acronyms, and definitions that will be used throughout the entire document.

- JPM is an acronym for Java Process Migration and is used to refer to this project.
- Java-PM is an acronym for Java-Process Migration and is used in the naming convention for the Java classes that make up the JPM solution.

- Java-PMP is an acronym for the Java-PM process.
- Java-PMD is an acronym for the Java-PM daemon.
- Java Virtual Machine will be referred to as the JVM.
- IPC is an acronym for Inter-process Communication.

2. Java-PM Process

2.1. Description

The Java-PM process is a process running in the JVM that extends the Java-PM abstract framework class. By extending the Java-PM abstract framework class, the Java-PM process can be controlled by the Java-PM daemon. The design of the Java-PM process framework allows for any Java process to “plug” into the Java-PM daemon and become “migratable”.

2.2. Logical Design

The power of the Java-PM process comes from the common framework that is implemented by all of the Java-PM processes to be migrated in a mobile computing environment. Figure 2.1 captures the essence of the communication between the Java-PM process and the Java-PM daemon that enables process migration.

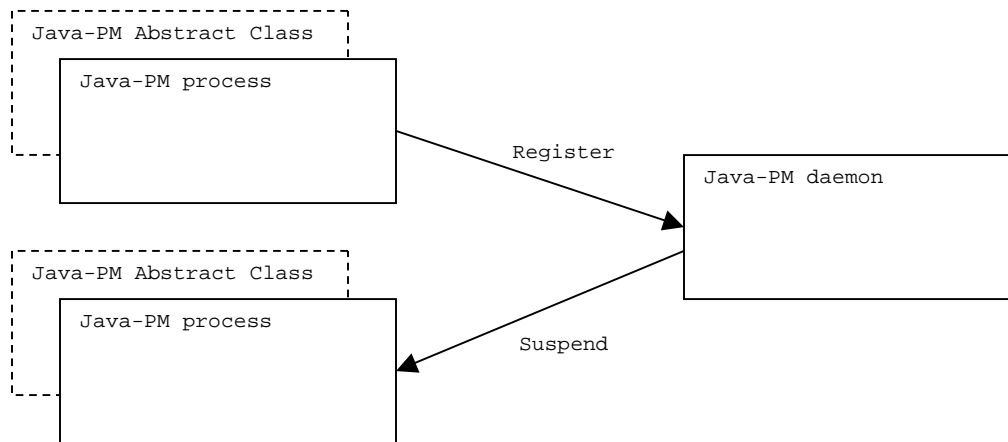


Figure 2.1

Figure 2.1 shows two processes interacting with the Java-PM daemon. The top process has just started and is registering itself with the Java-PM daemon so that it is known and can be considered for migration. The bottom Java-PM process has already registered with the Java-PM daemon and will be migrated. The Java-PM daemon is telling this process to suspend itself to prepare for migration.

Table 2.2 lists the interface that is exposed by the Java-PM abstract class. This is the interface that the Java-PM daemon will use to communicate with the Java-PM daemon.

Method Name	Description
-------------	-------------

Method Name	Description
public void register()	This method will handle the Java-PM process registration with the Java-PM daemon. This is the only communication between the Java-PM process and the Java-PM daemon that is initiated by the process. The Java-PM daemon will receive the process registration request and start to manage the process.
public JPMP_Status status()	This method will allow the Java-PM daemon to query the process to find out what it's processing status is to help make a decision to migrate this process. This process returns an instance of the Java-PMP_Status class. The Java-PMP_Status class is defined below.
public boolean suspend()	This method allows the Java-PM daemon to suspend a process so that it can be migrated. The details of suspended THIS process are implemented and hidden in this method. The Java-PM daemon does not have to know anything about the details of suspended any of the processes, just that the process is suspended.
public boolean resume()	This method allows the Java-PM daemon to resume a process once the process has been migrated to a node in the cluster. Again, the details of resuming a process are hidden from the Java-PM daemon. The Java-PM daemon is only interested in knowing that the process has resumed successfully.
public void complete()	This method allows the Java-PM daemon to stop managing the process and it also allows the process to complete any final processing on the original machine it started on. For example, if the process has output to display, this is the method where that logic would be handled. Once the Java-PM daemon calls this method on the process, the Java-PM daemon deregisters the process and stops managing it.

Table 2.2

Table 2.3 shows the definition of the JPMP_Status Java class that is returned from the Java-PM process status() framework method.

Data Attribute	Description
Percent_Complete	This data attribute holds the percent the Java-PM process is complete in its processing. The Java-PM daemon will use this datum to determine if migration makes sense for this process.
Process_Type	This data attribute holds the type of process this is. For example, is this process CPU intensive or IO intensive.
Data_Locality	This data attribute holds the locality description for the processes data. If the locality of the data is on another box, this process could very well be a great candidate for migration, etc.

Table 2.3

2.3. Physical Design

The communication between the Java-PM process and the Java-PM daemon can be implemented through several IPC options. File system IPC, sockets, events, or even Java RMI are all physical implementation options. A combination of sockets and events are most appealing for handling the communication between the Java-Pm process and the Java-PM daemon. This is mostly because of the locality of the Java-PM processes to the Java-PM daemon. The Java-PM process and the Java-PM daemon will be executing on the same machine, which makes an event driven IPC architecture favorable.

The Java-PM process will implement the Java-PM framework class and by doing this will have to implement the register() method. The abstract super class implementation of this method will handle most if not all of the gritty details of registering with the daemon on the local node.

A well known port will be established for all running Java-PM daemons. The Java-PM daemon will listen for Java-PM process requests on this port. This port number will be implemented in the Java-PM abstract framework class's register() method implementation. The Java-PM process will send it's process id to the Java-Pm daemon so that the daemon will be able to send the process signals.

Once the Java-PM process has registered, all further communication will be handled through events. The Java-PM process will have handlers implemented to receive these events and call one of the appropriate methods describe above. The event-driven paradigm being described here for IPC will be implemented through Linux signals. The proper Linux signal handlers will be implemented in the Java-PM process to "catch" the signals and proceed appropriately. The Java-PM daemon will send the signals to the Java-PM process accordingly.

2.4. Java-PM Process Sequence Diagrams

Table 2.4 shows the sequence of interaction between the Java-PM process and the Java-Pm daemon up to the process being serialized. After the Java-PM daemon has received the serialized process and has queued the process for migration, the process exits, which ends the interaction.

Java-PM process	Java-PM daemon
	Java-PM daemon Starts - The daemon starts, initializes, opens the port for communication, and begins to listen for Java-PM process registration requests.
Java-PM process Starts - The process registers itself with the daemon. The framework method register() attaches to the well known port and sends a registration request to the Java-PM daemon.	

Java-PM process	Java-PM daemon
	Java-PM daemon Receives Java-PM process Registration Request - Examines process by calling status(), sending the status signal to the process, on the process to determine if the process is "migratable".
Java-PM process receives the signal for the status command, calls the status framework method. The status method instantiates a Java-PM_Status object and returns it to the Java-PM process via the socket.	
	Java-PM daemon receives the instance of the processes Java_PM_Status object and examines it. The daemon determines if the process should be migrated. The Java-PM daemon calls suspend on the Java_PM process via a signal.
Java-PM process receives the signal to suspend itself and calls the suspend framework method. The process prepares itself for migration. Once suspension activity is complete, the serialized process is sent to the Java-PM daemon via socket communication. The Java-PM process then gracefully exits and stops execution.	
	The Java-PM daemon receives the serialized Java-PM process and queues the process for migration.

Table 2.4

Table 2.5 shows the communication between the Java-PM daemon on the node the process was migrated to and the resumed Java-PM process.

Java-PM process	Java-PM daemon
	Java-PM daemon on the remote node that has received the serialized Java-PM process, calls the resume framework method on the serialized Java-PM process.
Java-PM process receives the call to resume and resumes execution. When the execution is complete, the process sends a message to the Java-PM daemon that it is complete along with its serialized bits via sockets. The Java-PM process then gracefully exits and stops execution.	
	Java-PM daemon receives the message that the Java-PM process has completed execution and receives the Java-PM process serialized bits. The Java-PM daemon sends the Java-PM serialized process object back to the Java-PM daemon that owns this process.

Table 2.5

Table 2.6 shows the communication between the Java-PM daemon that owns the Java-PM process and the Java-PM process after the process has come back from the migrated node for execution.

Java-PM process	Java-PM daemon
-----------------	----------------

Java-PM process	Java-PM daemon
	Java-PM daemon calls the complete framework method on the serialized Java-PM process. The Java-PM daemon then deregisters this process and stops managing the process.
Java-PM process receives the call to complete. The process completes any final execution, display output or save a file etc. The Java-PM process then gracefully exits and stops execution.	

Table 2.6

3. Java-PM Daemon

3.1. Description

The Java-PM daemon is responsible for the management of the Java-PM processes that are to be migrated to different nodes in the cluster for process execution so that energy improvements can be realized. The Java-PM daemon handles the scheduling of processes to migrated, the logging of process to be migrated for efficient transmission, and the caching of processes. The Java-PM daemon is the heart of the system, communicating with Java-PM process locally and other Java-PM daemons within the cluster of nodes in the mobile computing environment.

3.2. Logical Design

The Java-PM daemon is made up of several components that work seamlessly together to migrate processes in the most efficient manner. Figure 3.1 shows the logical design of the Java-PM daemon and the different components that make it what it is.

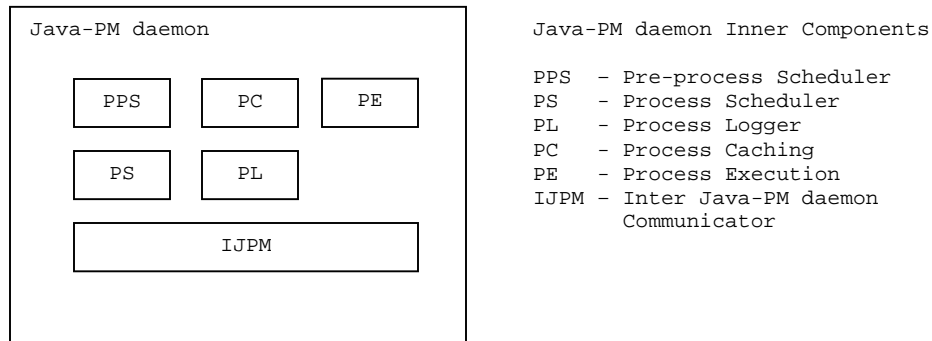


Figure 3.1

Table 3.2 describes the responsibilities of each of the components shown in Figure 3.1.

Java-PM daemon Component	Responsibilities
--------------------------	------------------

Java-PM daemon Component	Responsibilities
Pre-process Scheduler (PPS)	<p>The PPS component is responsible for handling the Java-PM process registration requests as well as determining if the process should even be migrated. This component will call the status framework method on the process and examine its information. When the PPS component has a process to migrate, it calls the suspend framework method on the process. In return the PPS receives the serialized process bits from the now stopped Java-PM process. The PPS is now finished working with the process and hands it off to the PS.</p> <p>The PPS component will interact with the IJPM component to communicate with the process.</p>
Process Scheduler (PS)	<p>The PS component is responsible for scheduling the processes in the most efficient way possible. The heart of the PS component is the algorithm it uses to schedule the processes.</p>
Process Logger (PL)	<p>The PL component is responsible for grouping process to be migrated in the most efficient manner for optimal transmission. If two process are being migrated to the same node in the cluster than send them together. This is the job of the PL component.</p> <p>The PL component will interact with the IJPM to figure out how to log the process together. When the PL component has a "process block" ready for transmission, the PL uses to IJPM to send it on its way.</p>
Process Caching (PC)	<p>The PC component is responsible for caching process locally so that if the same process is to be migrated to this node again, the process does not actually have to be transmitted.</p> <p>The PC component will interact with the IJPM to figure out if any processes are on their way that don't need to be sent.</p>
Process Execution (PE)	<p>The PE component is responsible for resuming a migrated process on the local node. The PE component also handles completion of a migrated process once a process has come back from the migrated node. The PE component will call the resume and complete framework methods on the Java-PM process accordingly.</p>
Inter Java-PM daemon Communicator (IJPM)	<p>The IJPM is responsible for all communication with the other Java-PM daemons on other nodes in the cluster as well as the local Java-PM processes. The IJPM encapsulate the logic necessary for IPC communication and exposes convenient methods for each of the components to do what they need to do.</p>

Table 3.2

Table 3.3 describes the interface the Java-PM daemon IJPM component exposes to the other Java-PM daemon components. The methods in this interface are built to meet the needs of each component.

Method Name	Description
public boolean send_pblock()	This method is called by the Process Scheduler (PS) component to send a block of processes to another node for execution. If the process block was successfully sent to the other daemon then this method sends a true value, indicating success, to the caller.

Table 3.3

Table 3.4 describes the RMI interface that is used for Inter Java-PM daemon communication.

Method Name	Description
public JPMD_Status status()	This RMI method is used by the Java-PM daemon to receive current information about the node the daemon is executing on. This method returns an instance of the JPMD_Status class.
public boolean send_pblock(pblock bytes)	This RMI method is used by a Java-PM daemon to send to another Java-PM daemon a process block. If the process plock is sent successfully, then a value of true is returned to the caller.
public boolean send_process(process bytes)	This RMI method is used by a Java-PM daemon to send back to the original node a process that has completed its execution on the migrated node.

Table 3.4

Table 3.5 shows the definition of the JPMD_Status Java class that is returned from the Java-PM daemon status() RMI method.

Data Attribute	Description
CPU_Utilization	This data attribute holds the data on how utilized the CPU is on this node.
Transmission_Cost	This data attribute holds the cost to transmit a process round trip to this node.
Cached_Processes	This data attribute holds the list of processes that have been cached on that node.

Table 3.5

3.3. Physical Design

To implement the Java-PM daemon communication, Java RMI will be used to take advantage of the ease of use in remote IPC communication provided

by Java RMI. A central and well known RMI registry will be established in the cluster of nodes that make up the mobile computing environment for the JPM project experimentation.

Each Java-PM daemon will register its services with the RMI registry. Each Java-PM daemon will have unique addressable registry service id known before execution of each experiment. Each unique Java-PM daemon registry service id will be known by all the Java-PM daemons executing on each node in the cluster of the mobile computing environment.

This, of course, does not model a real mobile computing environment and is a constraint on the experimentation performed. It has been decided that this will not affect the energy measurements from the experiments because once communication has been established between the nodes, the energy measurements can start to be taken.

When a Java-PM starts on a node, it queries each well known Java-PM daemon service in the RMI registry. When a service in the registry has been found, this node will then be on the list of nodes to potentially migrate processes to. At a certain frequency, the Java-PM daemon will query the RMI registry for Java-PM services that are not active. A node may go down and communication is lost with that node. The Java-PM daemon will certainly want to try to regain communication to that node when it comes back online. By polling the RMI registry, a Java-PM daemon will have a moderately active list of nodes that processes can be migrated to.

3.4. Sequence Diagrams

Table 3.6 shows the communication between the Java-PM daemons executing on nodes in the cluster of nodes in the mobile computing environment when starting execution.

Java-PM daemon on Node A	Java-PM daemon on Node B
The Java-PM daemon starts execution on Node A. The daemon registers its service with its well known and unique service id with the well known RMI register for this cluster of nodes.	
<p>The Java-PM daemon process examines the well known list of participating Java-PM daemons in the cluster of nodes and attempts to establish communication with each one.</p> <p>To establish communication with the other Java-PM daemons, the daemon will query the RMI Registry using the well known registry service id's of the other participating Java-PM daemons in the cluster of nodes.</p> <p>It is found that the Java-PM daemon executing on Node B has registered with the well known RMI registry.</p>	

Java-PM daemon on Node A	Java-PM daemon on Node B
The Java-PM daemon process calls the common Java-PM daemon RMI method <code>jpmd_status</code> to gather current information about the Java-PM daemon executing on node B.	
	The Java-PM daemon receives the method call from the daemon running on node A. This daemon's <code>jpmd_status</code> method is invoked and returns a <code>JPMD_Status</code> object to node A via RMI.
<p>The Java-PM daemon receives the <code>JMPD_Status</code> object from node B and examines its contents. Based on the data received from node B, node A decides whether to consider node B for process migration.</p> <p>After examining the status data from node B, node B is put on the list of nodes to migrate process to.</p>	

Table 3.6

Table 3.7 shows the communication between the Java-PM Daemon Process Scheduler (PS) component and the Java-PM Inter Java-PM Communicator (IJPM) component.

Java-PM daemon PS Component	Java-PM daemon IJPM Component
The Java-PM daemon PS component has a process block to migrate to another node on the cluster. The PS component invokes the IJPM component to do transmit this process block to the other nodes by calling <code>send_pblock</code> on the IJPM component.	
	The Java-PM daemon IJPM component receives the request to send the process block for migration. The IJPM examines the status of the other nodes in the cluster and determines which node to send the process block to. The IJPM migrates the process block successfully and sends a success message to the PS component.
The Java PM daemon PS component receives the success message from the IJPM component.	

Table 3.7