

Design and Implementation of Distributed Process Execution Environment

Project Report Phase 3

By
Bhagyalaxmi Bethala
Hemali Majithia
Shamit Patel

Problem Definition:

In this project, we will design and implement a distributed process execution environment. The system will have several sites, each having processes running. The site receiving a new task will first analyze its load and decide whether it can execute the task. If it decides to execute the task, it will locate and fetch the code (object) corresponding to the task using the Service Location Protocol. The site will then execute the code. However if the site is loaded and decides not to execute the new task then it will discover some other site, which is willing to execute that task. The new site is selected in a way that the system is load shared. The new site willing to execute the task will then locate the task, fetch it and execute it using load balancing algorithm.

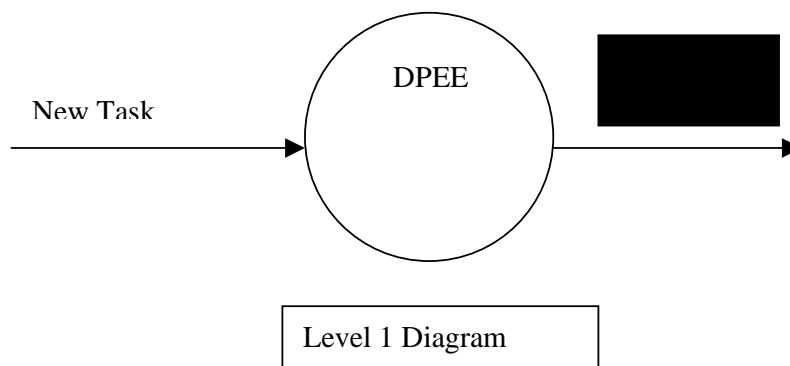
Assumptions:

The following assumptions are made for the system to be implemented:

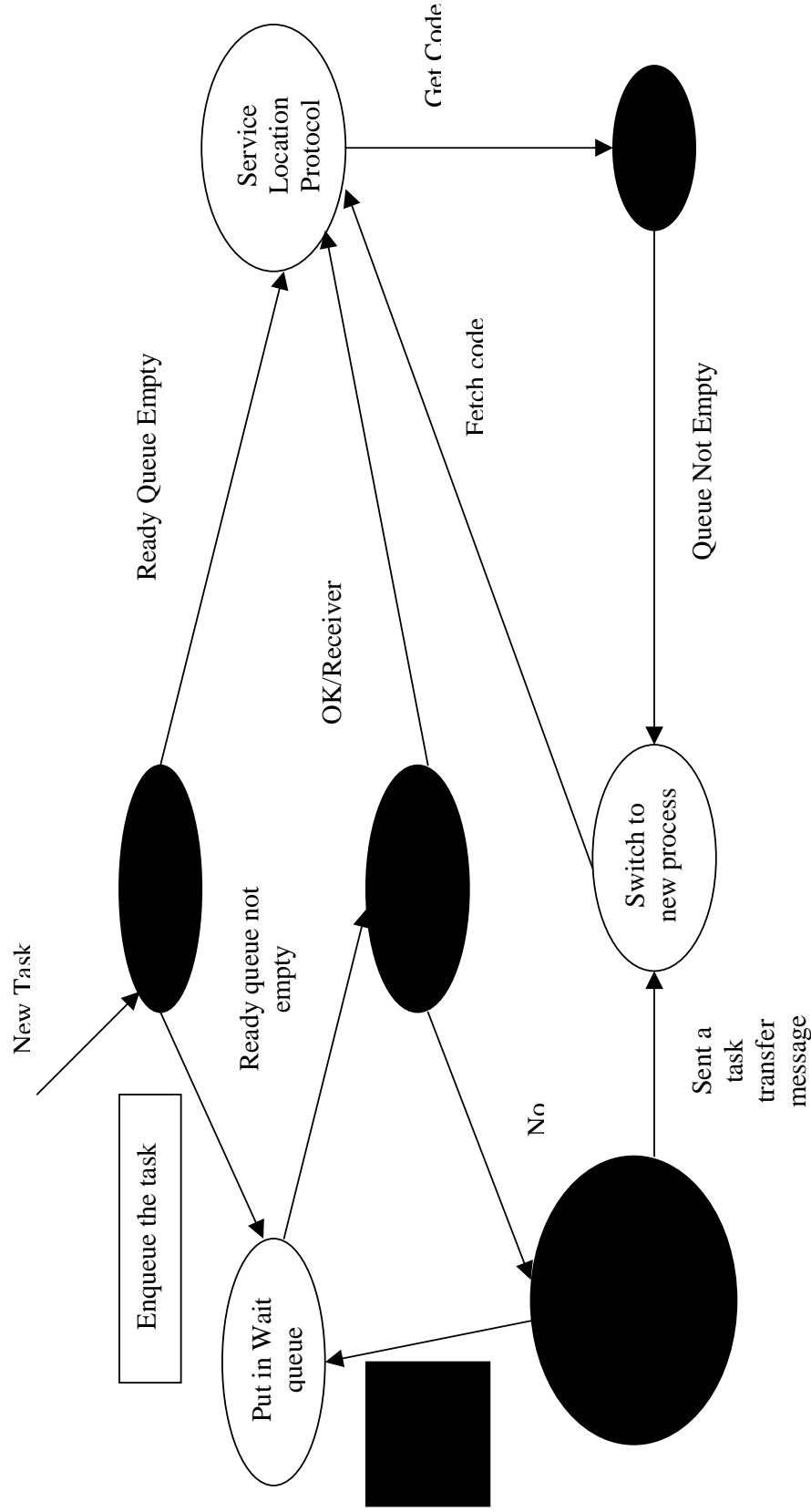
- 1) The system is dynamic i.e., new sites can be added to the system and none of the sites can go down.
- 2) Initially the sites will be idle i.e. will not be executing any tasks.
- 3) Two queues will be maintained for each site in the system; ready queue and wait queue. When a new task arrives at a particular site, it is put in the wait queue. When the site is ready to execute the task, the task is transferred from the wait queue to the ready queue.
- 4) The Service Location Protocol is used to locate the code to be executed for the task. The Service Location Protocol is localized at each site.
- 5) If the system is overloaded i.e. exceeds the threshold limit, the new tasks are discarded from the system.
- 6) FIFO scheduling is used for the queues in the system.
- 7) A modified version of Stable Sender's Algorithm is implemented for load balancing in the system.
- 8) The code of the task is available only at one particular site. Multiple copies of the code are not available.
- 9) We need to have a time interval between the addition of two new sites.

State Diagrams of the system:

Context Diagram

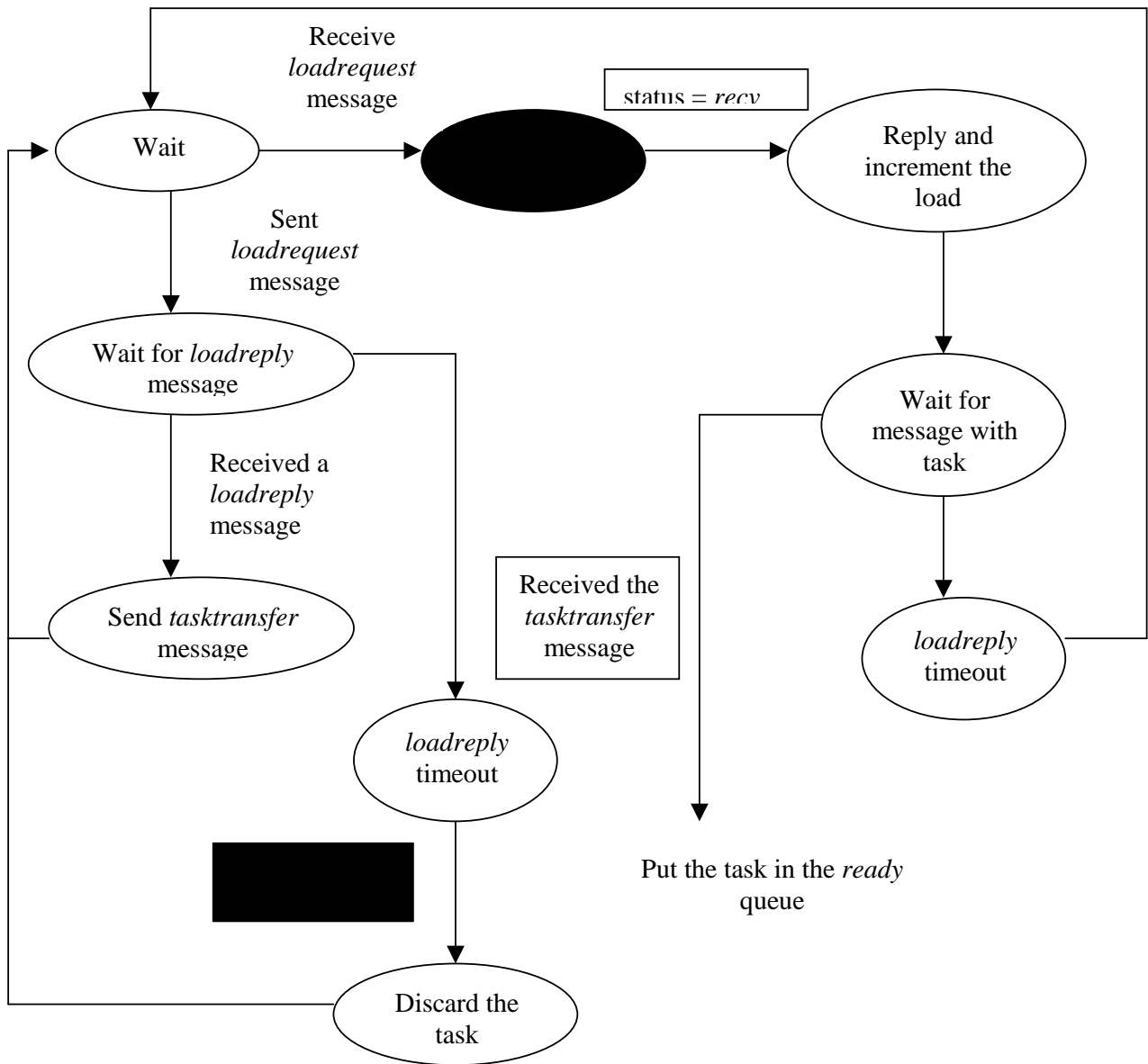


Level 1 State Diagram

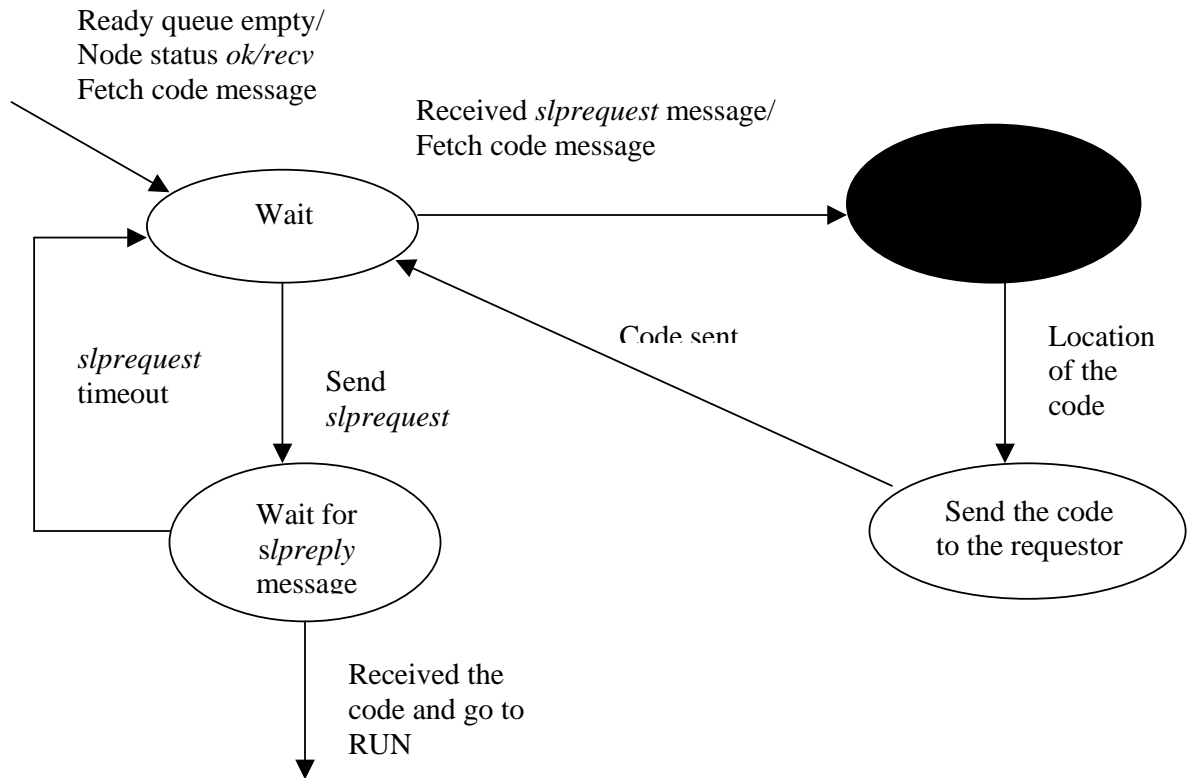


Load Balancing Algorithm

loadreply timed out



Service Location Protocol



Design Specifications

Each site maintains its status i.e. whether it is in a receiver, OK, or a sender state depending on the threshold. Threshold is decided by process queue length. Every site in an initial state has no processes in its ready queue. When a new task arrives, it is put in the wait queue and proceeds as follows:

(1) If the processor at the site is not busy executing some task or has finished executing a task, the wait queue is checked to see if it is empty. If the wait queue is not empty, then load status is checked and decision is made whether it can execute the new task or not. If it decides to execute the task (status is *ok/receiver*), the task is transferred to the ready queue. If site decides not to execute the task, the load-balancing algorithm is executed to find a new site where this task can be transferred. Once the site is determined and the wait queue is empty, it executes the next task from the ready queue.

(2) If the processor is busy, we put the task in the wait queue. Once the code is fetched, the task is executed. On completion of the execution, site proceeds as in (1).

Service Location Protocol (SLP)

This module is designed to locate and fetch the required code for the node. As assumed SLP is local to each site, it is invoked just before the execution of the task. The SLP service is provided to a site with the help of two ports where the process will listen for messages from other sites. One of the ports is used for receiving replies for this site and the other port for receiving messages from other sites. Whenever the service location protocol is invoked for fetching code by a site, it sends broadcast message, *slprequest* and waits for replies from other sites. The site that has the code for the program responds to the *slprequest* message with a *slpreply* message. Once the code is fetched, it goes back to listen at the port. Site can also get a *slprequest* from other sites that are trying to fetch some code. To handle this type of messages we use the second port. If a site has the code that is requested, it responds with a *slpreply* message and goes back to listen at that port for *slprequest* messages.

Load Balancing

The load balancing policy used is the sender-initiated algorithm.

Selection policy: Whenever the load of the site becomes greater than the threshold, it becomes a sender. The load of the site is a measure of the length of the Ready queue length.

Transfer policy: The new task that arrives at a site is chosen for transferring.

Location policy: The load-balancing algorithm uses two ports for distributing the overall system load. It works in the following way: we bind a process to each of the two ports and keep listening at the ports for any new messages. Whenever a site becomes a sender, it invokes the load-balancing algorithm. It sends out a unicast message, *loadrequest* from receiver list, in an attempt to find a receiver. It then sets up a timeout interval and waits for replies from receiver. If a node considers itself as a receiver, it replies with a *loadreply* message and increments its load by one in anticipation of the new incoming task. On receiving the *loadreply* message, the sender removes the task from its wait queue, and sends a *tasktransfer* message. On receiving a *tasktransfer* message, the receiver adds the new task to its ready queue. On the other hand if a site gets a *loadrequest* message it responds only if it is a receiver. If the site receiving a *loadrequest* message is not a receiver then it does not reply. Next site from the receiver list is tried until they become empty. When the receiver list becomes empty then sites from OK list are sent messages. If none of the sites from the OK list are receivers then the process is discarded. It thus implements modified sender' initiated algorithm.

Information policy: The information policy is demand driven policy. Whenever a site becomes a sender, then only it tries to update the status information.

Implementation

Modules

- 1) myDPE : This is the starting module of the system. An instance of newDPE is created in it. This module takes no inputs and does not return any value.
- 1) newDPE : This module initializes the new site in the system and integrates it with the existing sites in the system. It initializes the ready queue, wait queue, directory, loading balancing, service location and the addlist. (receiver list, sender list and ok list)
- 1) dirinit1 : This module handles the configuration of the site. This includes getting the port number for each site from the user, creation of directory, addition of processes to the site.
- 1) addlist : After the configuration of the new site, a broadcast message is send to all the other sites informing them about it's port number and IP address. The existing sites respond with their corresponding IP addresses and port number, which is added to the receiver list of the new site.
- 1) Lb1: This module takes care of load balancing in the system. We use the modified version of stable sender initiated algorithm. We use three queues i.e. send queue, receive queue and ok queue. Initially all sites in the system are assumed to be receivers. The receiver queue is checked first when the site is overloaded. The head of the queue is obtained to get the IP address and port number where the request is sent to transfer the task. If the site has not changed its status by then, the task has to be executed by the site. If the site in the receiver list is also loaded then it proceeds to check in the ok list and then the sender list. If no site is ready to execute the task, then it discards the task.
- 1) Localsearch1: Two functions are implemented in this module. Localfilesearch returns true if the process to be executed is present in the local site. If the process is not found then broadcast message is sent to all the other sites in the system. The site having the source file for the process to be executed appends the file to the reply message. The local site fetches the file from the packet and stores this file into its directory. Search and execute function compiles and executes the source code in the local site. After the execution the file, it is deleted from the local site.
- 1) Multicastmessage: This module is primarily responsible for multicasting messages and getting unicast replies.
- 1) Miscellaneous Modules (getuserport, getuserinput, q2, q): getuserport is mainly concern with getting the port numbers in the form of integers. q2 and q are queues implemented to adapt to the system.

Test Modules

a1,a2,a3 test1, test2, test3, test4, test5, test6, test7, test8, test10, test11, test12, test13, test14, test15, test16 : These test modules are used as processes to be executed in the system.

Strength of the System:

As seen from the load-balancing algorithm used, the strength system tries to minimize the network traffic. We try to find a receiver and then transfer the task rather than trying to find the best receiver. This improves the response time of the task that is transferred as we try to maintain the status of the system. As the information policy is demand driven, i.e. only when a site becomes a sender does it poll sites and either transfers the task or gets the current status of the polled site. This again helps in reducing the network traffic rather than informing all sites whenever the status is changed by broadcasting.

Another strength of the system is the dynamic configuring of the site with respect to the code that is available at the site. In order to make the availability of the code dynamic, at the time of bringing up the site, we present the user with a set of test programs, the code of which can be made available at that site.

Our system does not provide the user with a Graphical User Interface as it involves overhead on the system.

Drawbacks of the system:

- 1) Preemptive task transfer is not implemented.
- 2) Since the LAN was used, and since we are transmitting the file compiling, executing and deleting the local copy at the user machine we have not used serialization.
- 3) The lost of messages is not taken care of by the system.
- 4) We have not implemented any security measure for the system.
- 5) During load balancing implementation of the site, first the receiver list is checked for the sites to send messages. If receiver list is empty then it would be a better design issue to assume that the OK list and the sender list be queried as the last elements unlike the receiver list which is queried for get head.
- 6) There should be a time delay before the integration of two new sites to the system.
- 7) the response time of the task may increase as the system does not maintain the up to date status information of the system.

Packages used are

java.io
java.lang
java.net
java.util
javax.swing

Time Line:

We have followed time schedule to implement our system:

24th Oct - Phase 1

27th Oct - Pseudo Code of each Module

28th Oct - Start Coding

04th Nov - Service Location Protocol

08th Nov - Load Balancing Algorithm

09th Nov - Integrating SLP and Balancing Algorithm

13th Nov - Design Test Modules Implementation

21st Nov - Update the design and make the next design document

01st Dec - Testing

07th Dec - Document the entire report

12th Dec - Submit the final report and give demo.