ITU-PRP: Parallel and Distributed Computing Middleware for Java Developers

Enis Spahi
Istanbul Technical University, enisspahi@gmail.com

D. Turgay Altilar
Istanbul Technical University, altilar@itu.edu.tr

ITU-PRP: Parallel and Distributed Computing Middleware for Java Developers

Enis Spahi[1], D. Turgay Altılar[2]

[1] Department of Computer Science, Istanbul Technical University, Istanbul, Turkey
enisspahi@gmail.com[1], altilar@itu.edu.tr[2]

Abstract. ITU-PRP provides a Parallel Programming Framework for Java Developers on which they can adapt their sequential application code to operate on a distributed multi-host parallel environment. Developers would implement parallel models, such as Loop Parallelism, Divide and Conquer, Master-Slave and Fork-Join by the help of an API Library provided under framework. Produced parallel applications would be submitted to a middleware called Parallel Running Platform (PRP), on which parallel resources for parallel processing are being organized and performed. The middleware creates Task Plans (TP) according to application's parallel model, assigns best available resource Hosts, in order to perform fast parallel processing. Task Plans will be created dynamically in real time according to resources actual utilization status or availability, instead of predefined/preconfigured task plans. ITU-PRP achieves better efficiency on parallel processing over big data sets and distributes divided base data to multiple hosts to be operated by Coarse-Grained parallelism. According to this model distributed parallel tasks would operate independently with minimal interaction until processing ends.

Keywords: Parallel computing, distributed computing, java, ITU-PRP

1 Introduction

ITU-PRP provides an all-in-one solution for Parallel Programmers, with a Parallel Programming Framework and a Task Execution Middleware within a single system. ITU-PRP intends a simple way for Parallel Application Development, which makes Parallel Code easy to implement through a Java Library released as JAR Package. The regarded library contains implementable interfaces, which would generate autonomous parallel tasks written as sequential code blocks. Parallel tasks are operated according to Loop Parallelism and Divide and Conquer parallel models [1]. Additionally, ITU-PRP's distributed middleware provides resources for parallel processing and ensures execution of tasks. Computing resources are assigned dynamically according to System's real time conditions.
Parallel Programming Framework mostly encapsulates parallel operations and provides abstraction to developer. Multi-Host parallel operations are handled by the encapsulated package. Developer will not deal with Parallel Task Distribution, Task Execution, Task Reunification, Result Collection, Synchronization and Connection issues. Only some initial parameters regarding to task execution are required to be set as configuration on the implemented code. User will configure his application on the regarded platform with parameters specified for parallel task execution. Any user submits its produced applications for future task execution, request for task execution and collect execution results. Submitted applications are treated as tasks in the system, so once an application is submitted to system, it will be named as Task.
Users with their accounts for ITU-PRP System will connect to system through a web based graphical user interface. This web application would serve users for their operations on ITU-PRP System, especially on Task Execution Middleware. Authenticated user initializes task execution and views the results of parallel processing through a specified screen.
ITU-PRP expects contribution in terms of execution resources from any user using the platform. Any user logged in to ITU-PRP will be considered as potential resource. Connected clients are registered as Hosts as well, in order to make possible serving other Task Execution Requestor clients. Hosts will be available as potential computational resources during their idle times. Considering that many computers are mostly idle, the approach of this research has been utilization of non-used executional power in order to achieve high performance parallel applications.

2 Related Work and Motivation

Parallel API Interfaces like Message Passing Interface (MPI), Parallel Virtual Machine (PVM), OpenMP [2] implementable in native languages (C/C++, Fortran) are well known and used parallel frameworks for parallel processing. They achieve high performance on parallel processing on multi-core environments. But the fact that such systems are used on low level programming languages, developing distributed parallel systems is harder than any high level programming language. Also, some parallel Java implementations have been developed and become widespread within last years. Parallel Java implementations would be classified under three categories, API Interfaces derived from native interfaces, API Interfaces derived from native thread models and Distributed Parallel Systems. In this research, the focal approach is on the third category.
API interfaces derived from native interfaces, involve API interfaces derived from native C/C++, Fortran interfaces like MPI [2], PVM. Wrappers over MPI, PVM implement directives, provide adapted implementations. jPVM, MPJ-Express [3], Java MPI [4] are some of the existing ones.
API interfaces derived from Java native thread models, involve interface models developed from Java native Threads and communication protocols provided by Java. JOMP, and JaMP [5] API interfaces have their directives adopted from OpenMP. JADE (Java Agent Development Framework) [6] as another specific Framework implemented on Java, provides a framework for Parallel Processing.
Distributed parallel systems, involve Network Based technologies utilized to use distributed parallel resource. One some implementations, embedded Java Applets on Web based applications, create processes running on Client computers. Applets are downloaded from the regarded URL to browser's cache during first initialization and will behave like applications installed on Client's computer. Systems like Javelin [7][8], JAVM [9] implement Java in order to use computational resource on a wide range network over Internet.
Arguments that motivated us for developing ITU-PRP commonly with other existing systems are as follows:

- Utilizing hosts as potential Computational Power
- Implementations based on Java Applets
- Platform Independency, ”write once, run anywhere” philosophy
- Automatic Parallelization

Arguments that motivated and made us excited about ITU-PRP study, with specific features are:

- Computational hosts communicating each other via Peer-to-Peer protocol
- A pre-prepared object oriented pattern for Automatic Parallelization
- Strengthen scalability by applet based architecture
- Computational resource management by a special scoring system

3 ITU-PRP Design and Architecture

ITU-PRP system is designed as a web based system, which mainly utilizes Java Based Applet technology and does parallel processing operations on user's Web Browser. Prior to system log on, initialized Applets processes gather user information and are registered to Task Execution Middleware. The Process and Threads created on user's process behave as Hosts during their idle states and wait assignments of a task execution. If user requests for Task Execution, main process will behave as Client and do operations accordingly. Fig. 1 illustrates two main entities of system architecture; Parallel Programming Framework and Task Execution Middleware.

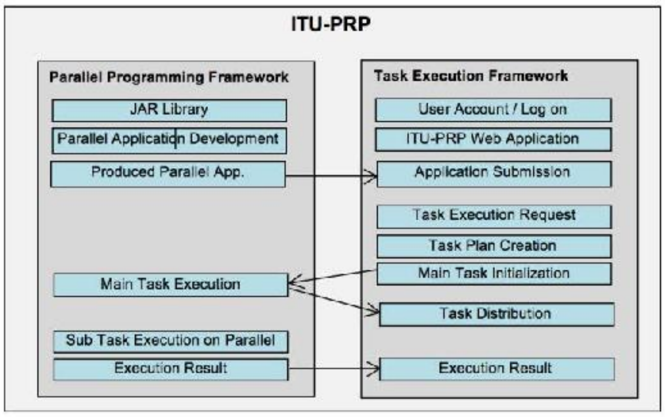


Fig. 1. ITU-PRP Framework Services

ITU-PRP System is designed with the goal to provide users an all-in-one platform with separate self-functioning components integrated for a single purpose, realizing high performance parallel execution by easy implementation. The regarded system components defined as Broker, Client and Hosts operate within an integrated Web Based System. Initially, user logs on to a web based application, on which a Java Applet is initialized and creates a process along with some set of threads for parallel processing. Broker as the coordinator entity, manages Hosts and Clients' activities.
While Broker is hosted on a Web Servlet, Client and Hosts work on a Web Browser, which would make this Framework widely usable without installing any additional application on Client or Host's computer. Application will work on any Java enabled Web Browser.

3.1 Parallel Programming Framework

Parallel Programming Framework basically provides parallel code blocks for application developers. User adapts his application code to patterns specified by this framework. Provided JAR Package is named as Parallel Programming Library, which is implementable by the user according to specifications on Implementation Guidelines. The produced application will be uploaded to ITU-PRP Web Application, which is a unit of Task Execution Middleware. Besides the implementation, execution of Main Task, Parallelized Sub Task Execution and result generation are background operations hided from the user.
The essential concerns on the design of the Parallel Programming Framework:

- **Easy Implementation and Simplicity:** Provide an easy to implement Parallel Programming Library to developers even not familiar with parallel programming.
- **Scalability:** Regardless of parallel code running on 2 or 10 hosts, written code should be same. Number of hosts is a configuration issue on Task Execution Middleware
- **Performance:** Caching mechanism of Java during Applet execution also makes application execution faster after first time execution.

3.2 Task Execution Middleware

Once an Application is submitted to Parallel Running Framework, it becomes a registered application on the repository. Authorized users are able to request execution for their application. Client's Task execution requests are processed on Broker. A Task Execution Plan with assigned hosts is provided for multi-host parallel execution. Once Client gets the result plan, it executes the main task of the application under its Java Applet process. Client's main task will distribute parameter information to assigned Host's by communicating on peer-to-peer protocol. This step is characterized as Task

Distribution phase and is made in parallel. After the finalization of task executions, main task will respond with the result of Execution to Framework.

The essential concerns on the design of Task Execution Middleware:

- **Security:** For security reasons, user is restricted to execute, check the results only for his or modify only his own task execution requests. Other users serving as Hosts during their idle times would notice some activity on their Java Applet processes, but calculated data and results are hided, unless the owner of application has put some output logs during development of it.
- **Performance:** Broker predicts behavior of Hosts in terms performance and network delays. Information like IP Addresses, Country, City, Location Info, Response Time, intensity, CPU and Configuration information are inspected for this purpose. Broker does consider this information, in order to prepare the best available Task Execution Plan to the Client.

Interaction of Clients, Host and a Broker are illustrated in Fig. 2.

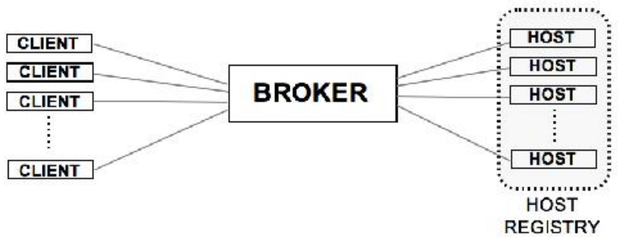


Fig. 2. ITU-PRP Design.

3.2.1 Client

Client requesting for Parallel execution gets Task Plan with assigned resources for Task Execution. Client initiates the Main Task, creates Sub Tasks and divides data to each one. Client creates Threads on behalf of subtasks and each Thread waits for the calculation results distributed to other hosts. Clients and Hosts communicate with Peer-to-Peer socket.

3.2.2 Host

Hosts are registered on Host Registry during the initial connection to System. IP Addresses, Country, City, Location Info, Response Time, intensity, CPU and Configuration information of Hosts are saved on Host Registry in order to be considered for decision purposes during Task Plan creation phase. This information will be retrieved by Broker and saved to Host Registry, in this case there Host will not be sending its information to Broker which will reduce overhead on Host.

3.2.3 Broker

Broker's responsibility is serving Client's managing resource Hosts. Broker provides Task Plans to Client's requesting parallel processing, with assigned Host Resources. On the other hand, Broker registers Hosts and creates records for each one. Records are added to Host Registry, which is characterized as collection of available resources. Records on Host Registry may behave both as Client or Hosts depending on the activity status of the Client. If a Host is on the state of requesting an execution plan, acts as Client. In case of idle state, it acts as Host available for Parallel Execution resource for other requestor Clients. Also, Hosts leaving the system are removed from Host Registry. Broker may refuse request of Client, in case of non-sufficient available resources. Broker is designed to be a Java Servlet running on a web server.

3.3 Host Registry

Registered Hosts on Host Registry go through some information retrieval phases during their lifetime. Threads collect information periodically, as given in Table 1.
Broker updates information of hosts and modifies them on Host Registry. Within all these ones, Response Time is one of the most important ones. In order to perform information retrieval, Broker sends special PING messages to any Host and expects a PONG message. On the other side, Host Listener Thread listens for incoming Ping messages from Broker and responds with a Pong message accordingly. The interval between Ping and Pong messages gives the response time of communication between Broker and Host.

Table 1. **Record On Host Registry.**

Information	Explanation	Example
Host Name	Information generated by Host during Host registration	*26951e56-3b18-4efd-82de-ef2c3f339b8d*
User	User Name of the Client or Host	*Genericuser*
IP Address	Retrieved IP Address of the Host	127.0.0.1
Response Time	Renewed Periodically through PING / PONG messages	*1 ms*
Free Memory	Free memory declared by Host	*111720208 (Byte)*
Available Processors	Available Processors of Host	*4 (core)*
Active	Host's status for availability	*true/false*

Records on Host Registry have three potential states, such as Client, Available Host and Busy Host. By default user's process remains on Available state, unless he requests for a Task Execution or serves for execution to other Clients. Client to Host state transitions are occurred as illustrated in Fig. 3.

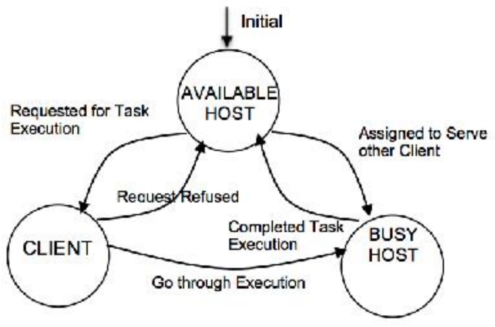


Fig. 3. Client and Host State Transitions.

Two possible cases triggers status change to Client or Busy Host states. In case of user requesting an execution the state will be transformed to Client state. Alternative transition is Busy Host state, which is occurred in case of Broker assigning the Host for serving any Client. As the Host completes its task process, it will return to its original State, Available Host. Also, Client doing parallel task execution will be transformed to Busy Host during the execution and return to original Available Host state finally.

3.4 Task (Application) Repository

Task Repository consists entries of registered parallel applications. Broker creates entry for JAR packaged applications and adds to Task Repository. Applications will be available for execution on Client and Hosts. Registered entries of repository keep information as showed on Table 2.

Table 2. **Entry On Task Repository.**

Information	Explanation	Example
Application ID	A unique key assigned by the broker from a sequence.	*37*
Parallel Task Name	Task name defined by the user. This is the identity name of the task on ITU-PRP web application.	*Parallel Sums*
Main Task Name	Main task name defined for Reflection API to initialize the Main Task on Client Applet.	*com.itu.ppp.examples.SumsWithFuture*
Callable Task Name	Task name defined for Reflection API to execute the implementation code of sub-task on Host Applet	*com.itu.ppp.examples.Sum*
Required Host Count	Required host count for parallel processing. This value is defined by user.	*4*
Parallel Task JAR URL	The URL Path of the JAR application uploaded by the user.	*http://parallelpattern:8080/AppletParallelProgImplementation/jars/ParallelProgram.jar*

Parallel Task Version	Version number of the application submitted.	*1.00*
User	Owner of the application which will be authorized to execute his Task	*genericuser*

3.5 Task Execution

Task Execution involves a set of operations under Task Execution Middleware, in order to complete parallel processing. Initially, users do request for execution of their Applications. Then Broker would respond to requests with a set of assigned resource hosts. The decision for assigning hosts is made according to a scoring mechanism performed by the Broker. As a result, high rated available hosts are provided to requestor Clients. In the meantime Client will be responsible for initiating the main task, distributing the fragmented data to sub-tasks, sending divided data sets to each task and collecting back after each Hosts execution is finalized. Data messaging between Client and Hosts are made via peer-to-peer protocol instead of a centralized protocol. On the other hand, idle Hosts, which are on the Available Host state, have their dedicated Listener Threads, which wait for incoming Task assignments. Both Client and Hosts download and cache packaged JAR application from the Task Repository during execution. Java Reflection API, Remote Class Loading and Object Serialization are the technologies implemented for these purposes. Also, Remote Jar Packages executed under the Context of Java Applet, are cached and executed from the local cache unless the JAR is modified or the Cache is cleared forcefully.

3.5.1 Host Resource Request

Client's which request for Host Resources contact Broker Service to get Task Execution Plan for parallel operation. The set of activities performed during Host Resource Request are shown on the Sequence Diagram illustrated in Fig. 4.

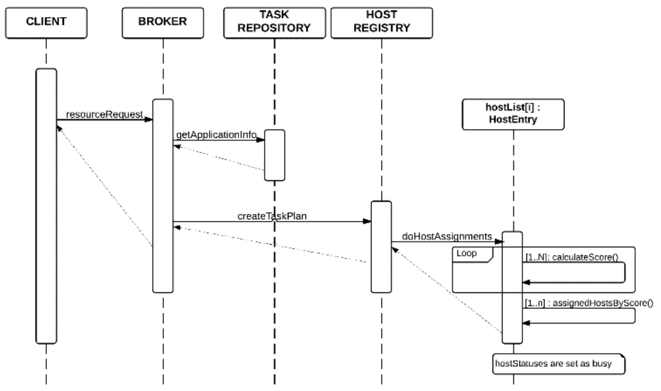


Fig. 4. Activity During Host Resource Request.

At first step, Client asks Broker to provide a Task Execution Plan for performing Parallel Processing. On next step, Host Registry assigns requested number of host resources according application information on Task Repository (host count, main task name, callable task name, Task URL). Assignment of resources is made by a scoring mechanism and Task Plan will be generated as result for Host Resource Request.

3.5.2 Scoring for Host Selection

Broker should provide the best possible resource in order to achieve worth parallel performance over sequential performance. A scoring algorithm, performed for Host Registry records does this. Information like Client's Location, Host's Location, Host Response Time, Free Memory and number of CPU cores are being considered to calculate cost based scores. The Hosts with lowest costs are being selected.

Host Response Time, which is the measured as time difference between Ping and Pong messages is an important parameter about how fast may a Host respond to a task assignment within a distributed network. Also, other additional information regarding Host's computational power, which are Free Memory and number of CPU cores are other considerations during scoring operation. The calculated cost values for each host are compared to each other and the lowest ones are picked and provided for Task Plan creation.

3.5.3 Task Plan

Task Plan, which is generated as a response for Host Resource Request, is structured from a list of assigned Resource Hosts and is provided to a Requestor Client. Length of Resource Host entries within Task Plan would be number of Parallel CPU's doing the Task Execution for Parallel Processing. Client will distribute Sub-Tasks to each Host provided on Task Plan and will initiate parallel processing.

Table 3 gives an example of a Task Plan provided by Broker to Client. Host Address contains IP Address and port number of Resources Host, to which Client will connect and notify for an execution request. Host Name is the unique id of assigned Hosts. On the other hand, JAR URL and Callable Task Name is sent to Host to specify which application and function will be executed by the Host.

Table 3. **Task Plan Example.**

Resource Host	Resource Detail
1	**Host Address:** 192.168.56.103:2049 **Host Name:** b4b45ff2-04e3-4af5-b1af-19d62d711807 **JAR URL:** http://parallelpattern:8080/AppletParallelProgImplementation/jars/ParallelProgram.jar **Callable Task Name:** com.itu.ppp.examples.Sum
2	……….
3	……….

4	**Host Address:** 192.168.56.102:2049 **Host Name:** 2496f352-260e-4799-8790-8eaea4b7109b **JAR URL:** http://parallelpattern:8080/AppletParallelProgImplementation/jars/ParallelProgram.jar **Callable Task Name:** com.itu.ppp.examples.Sum

3.6 Parallel Processing

Client Applet, which requests for the execution of an Application will act according to Task Execution Plan. Client communicates Resource Hosts via given IP Addresses. By the finalization of the Task, Client sends a Task Execution Report to Broker. Steps performed during the Task Execution are illustrated in Fig. 5, on which Main Task, Sub Tasks, Thread Pool and Hosts are the performers of the Task Execution cycle.

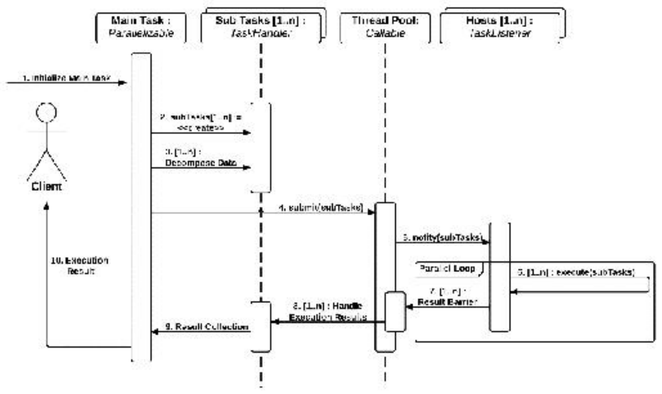

Main Task : Parallelizable
Sub Tasks [1..n] : TaskHandler
Thread Pool: Callable
Hosts [1..n] : TaskListener
1. Initialize Main Task
Client
2. subTasks[1..n] = <<create>>
3. [1..n] : Decompose Data
4. submit(subTasks)
5. notify(subTasks)
Parallel Loop
6. [1..n] : execute(subTasks)
7. [1..n] : Result Barrier
8. [1..n] : Handle Execution Results
9. Result Collection
10. Execution Result

Fig. 5. Parallel Processing Steps

Parallel Applications are developed according to implementation pattern of Parallel Programming Library. Developer implements Main Task and Sub Task code blocks on which he specifies the work of Client and Hosts. While Main Task is created and executed on the Client, Sub Task is created on the Client but performed on the Host. Steps on Parallel Processing cycle are follows.

1. Initialization of the Main Task: Application's Main Task is initiated on Client Applet. Main Task is an implementation of Parallelizable interface.

2. Create a Sub Task Object for each available host: Developer creates Sub Task, within the scope of Main Task. Sub Task is an implementation of TaskHandler.

3. Decompose Data into available number of hosts, set Data to Sub Task Objects: Sub Task's data is set during creation of Sub Tasks. Shared memory is also applicable by setting same full data set to all Sub Tasks.

4. Create a Thread Executor Pool, Submit Sub Tasks into Thread Executor Pool: Thread Pool mechanism of Java Concurrent API is utilized for Thread based operations within Main Task. An ExecutorService with a length of available resource Hosts count is initialized.

5. Notify each Host for Task Processing by sending the Sub Task to each one: Sub Tasks are submitted to executor service. A notification service sends Application's JAR URL, Task Name and Serialized Object Stream of the Sub Task Resource Host's Task Listener Thread.

6. Sub Task Execution on each Host: Host's Task Listener Thread which gets an incoming notification from a Client, loads the regarded application from JAR URL and executes calculate function of the Sub Task and finishes processing. Task Listener sends the result back to Client by a Serialized Object Stream of the Sub Task.

7. Wait until Parallel Processing is completed on each notified Host: Sub Task object with its result field set after the execution on Host, is sent back to Host Notification Service. Executor Service ensures all Sub Tasks to be retrieved from Hosts by blocking the Main Task until all Sub Tasks are being processed.

8. Handle Execution Results: Sub Task results are retrieved from Executor Service.

9. Result Collection: Developer merges Sub Task results. Developer may do any manipulation on result collection (sum, average, etc) accordingly.

10. Provide Execution Result to Client: As final step, Main Task returns final result to Client and shows the result to user. Parallel execution is finished, all participants of execution plan are switched to its original states (Available Host).

4 ITU-PRP Implementation

ITU PRP's Parallel Programming Framework ensures abstraction of parallel processing via specified implementation patterns. Main Task is specified by implementing runMainTask method of Parallelizable interface. On the other hand, Sub Task is specified by implementing calculate method of TaskHandler abstract class. Data sets of Sub Task are set by the developer for processing on Hosts.

Program Code, Main Task Parallelizable interface (Parallel Programming Framework).

```
@Override
public String runMainTask(List<String> availHostAddr,String
                  parallelClassName, String jarURLAddress) {
  int availHosts = availableHostAddresses.size();
  ExecutorService executor =
        Executors.newFixedThreadPool(availHosts);
  List<Future<Long>> list = new ArrayList<Future<Long>>();
  for (int i = 0; i < availHosts; i++){
    Sum sum = new Sum(min, max); //Initialize Sub Task
    sum.setTaskConfig(availHostAddr.get(i%availHosts),
                    parallelClassName, jarURLAddress);
    Callable<Long> summing = sum;
    Future<Long> submit = executor.submit(summing);
    list.add(submit);
  }
  long result = 0;
  for (Future<Long> future : list)
  {result += future.get();}
  executor.shutdown();  //Wait result
```

```
    return String.valueOf(result);
  }
```

Program Code, Sub Task Handler Implementation (Parallel Programming Framework).

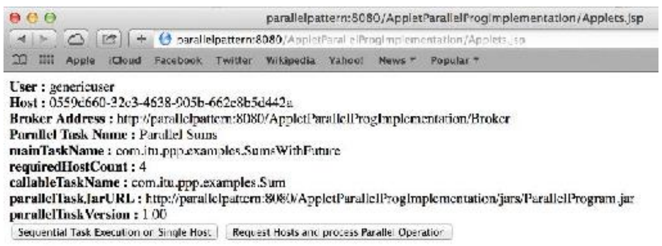
```
  @Override
  public void calculate() {//SUB TASK
    setResult(0); //INITIAL RESULT
    for (long i = from; i <= to; i++)
    {
      setResult(getResult() + i);
    }
    System.out.println(getResult());//FINAL RESULT
  }
```

ITU PRP operations like User Subscription, Client Logon, Application Upload and Modification on Repository, Task Execution are made through a Web Application designed in the System. ITU PRP Web Application is hosted on an Apache Tomcat Web Server located on the same location with Broker and P2P Server.

As the user logs on to Web Application, a Java Applet embedded to the web page will initialize and run. The regarded Java Applet creates a Host Listener thread, which will process as an available Host for the system on Client's computer.

User is able to view and select task on his repository with uploaded applications. Application submission to system is made through a page, on which user fills Application information like Application/Task name, Java class name with package hierarchy, suggested host count for execution, application version which will be considered base information for Task Execution. User may also select his application from repository and modify its information.

In case of selection of the task from the list of repository, the user will view a screen as shown in Fig. 6 Task information is viewed on the screen. User triggers may execute of the task Sequentially or in Parallel according to Task Plan proveded by Broker.


parallelpattern:8080/AppletParallelProgImplementation/Applets.jsp
parallelpattern:8080/AppletParallelProgImplementation/Applets.jsp
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular

User : genericuser
Host : 0559d660-32c3-4638-905b-662c8b5d442a
Broker Address : http://parallelpattern:8080/AppletParallelProgImplementation/Broker
Parallel Task Name : Parallel Sums
mainTaskName : com.itu.ppp.examples.SumsWithFuture
requiredHostCount : 4
callableTaskName : com.itu.ppp.examples.Sum
parallelTask.JarURL : http://parallelpattern:8080/AppletParallelProgImplementation/jars/ParallelProgram.jar
parallelTaskVersion : 1.00

Sequential Task Execution or Single Host
Request Hosts and process Parallel Operation

Fig. 6. Task Execution in ITU-PRP Web Application.

A Java library is provided to Parallel Developers to adapt their application codes for Parallel Running Platform. ParallelPatternFramework.jar can be downloaded from ITU PRP Web Site. A parallel developer must include the provided library file to Java project and implement its program code according to specifications. Detailed information on implementation may be found on ITU-PRP Web Site. Result implementation will be uploaded to ITU PRP Web Site to application repository. Result application should be packaged as Jar file as well. Application developer is required to fill Application/Task name, Java class name with package hierarchy, suggested host count for execution, application version. Library also hides task distribution and result collection operations from user in order to reduce complexity on developer’s point of view.

5 Experimental Results

Performance review is made over a typical parallel processing example, sum of the number of an array with big length of data. Considering objective of optimizing processing times over big sets of data, ITU-PRP’s Coarse-Grained parallel model would be applicable to such a case. Calculation base is divided to smaller units, on which processing on each part is made independently, processed on multi-host parallel tasks.

Vector summation (sum of the number sequence between 1 to n) is used as test scenario in order to compare performance measures. Measures are made in three different execution plans, sequential processing, multi-threaded processing and multi-host parallel processing. Test Applications for three cases, were implemented on Java. The first implementation designed as single threaded sequential application, did take 1,269ms long. The test was made on a computer with Intel i5 Dual Core 2.5 GHz CPU. According to second implementation, the base data set has been divided into four units to be processed on four independent threads, of which each one did calculate a sum of 100,000,000 numbers, then a main thread collected results of each thread’s calculation. The processing took long 1,130ms long, which means 10.95% processing time reduction and 10.95 speed up rate. Distribution of threads to CPU’s is in control of JVM and there is no information if JVM has sent four threads to two cores equally. Therefore, there is no guarantee on equal distribution to CPU Cores on Java native thread implementation. On the other ITU-PRP’s ITU-PRP did achieve a noticeable performance gain with 423ms total processing time, which means 66.67% of processing time reduction, 3,00 speed up rate over sequential processing. Execution was performed by an initiator Client dividing main data set to four units, which were distributed equally as Sub Tasks to four Resource Hosts, each one with Intel i5 Dual Core 2.5 GHz CPU’s. The Hosts are four Virtual Hosts located on a single Computer. Performance comparison is given in Table 4.

Table 4. **Test Results.**

Application Model	Hosts	Data Units	Process Time(ms)	Speed Up	Perf. Gain (%)
Sequential Java Application	1	1	1,269	1.00	-
Multi-Threaded Java App.	1	4	1,130	1.12	10.95
ITU-PRP Parallel Task Execution	4	4	423	3.00	66.67

The fact that four Hosts were virtual hosts, on an environment with minimal network connection delay times, brought such a high rate of speedup (3.00). On the other hand, multi-host experiments for hosts on a network with heterogeneous platform would cause additional delays. In this case, the speedup would vary depending on heterogeneity and interconnectivity quality. In order to estimate the impact of potential delay times, some ping tests within our network lab have been performed. Ping tests with 1MB data length took 5-10ms to achieve the destination hosts. The worst case delay time was calculated as 20ms for a round trip client-host connection. This delay would impact our 423ms processing time with an additional 20ms delay time, which gives an expected 443ms total processing time. In this case the expected speed-up will be 2.86, which is a reasonable rate as well. Another fact on delay reduction would be usage frequency and number of available volunteer hosts as well. This would give more options on selecting optimal hosts.

6 Conclusion

The methodology of Code Parallelization is based on Loop Parallelism, Divide and Conquer, Master-Slave and Fork-Join parallel models. ITU-PRP's design on Parallel Processing, in which Main Task creates and sends Sub Tasks to Hosts in a Loop Parallelized mechanism, aims to provide an object oriented pattern to combine with parallel models. Object oriented pattern and adaptability of this design is also another noticeable feature, compared to conventional native parallel development tools.ITU-PRP's approaches on Data Parallelization also provide some other noticeable features, such as customizing process on divide and conquer or shared memory models. Data distribution via Sub Task object serialization ensures users control over data parallelization. Object-based data distribution, instead of message-based distribution is also another feature, which provides flexibility to user to specify data types for distribution. Experimental results have revealed that multiple executions of application have speedup the execution due to Java Applet caching mechanism. A drawback on performance would be applications running first time would work slower, until are cached. But considering that parallel processing is supposed to be made multiple times over hosts, higher performance rates would be achieved by time. Statistics for Parallel Application executions and performance measures will be saved and logged. Any applications statistical information for their recent activity in terms of performance measures would be considered for future executions, so that higher utilization can be achieved on future execution plans.

References

1. Thomas Rauber, Gudula Rünger (2010), Parallel Programming for Multicore and Cluster Systems. 2nd Edition, Springer Heidelberg Dordrecht, London, New York
2. Michael J. Quinn (2004), Parallel Programming in C with MPI and OpenMP, 1st Edition, McGrawHill, New York MPJ Express: An implementation of MPI in Java, Windows User Guide, http://mpj-express.org/docs/guides/windowsguide.pdf javampi, Java implementation of object-oriented representation of MPI-1 Standart, https://code.google.com/p/javampi/
3. M. Klemm, M. Bezold, R. Valdema and M. Philippsen, (2007) : JaMP: An Implementation of OpenMP for a Java DSM, University of Erlangen-Nuremberg, Computer Science Department, Erlangen, Germany Jade, JAVA Agent Development Framework, http://jade.tilab.com
4. Peter Cappello, Bernd Christiansen, Mihai F. Ionescu, Michael O. Neary, Klaus E. Schauser and Daniel Wu, 1997: Javelin: Internet-Based Parallel Computing Using Java, University of Bristol, Department of Computer Science, University of California, Santa Barbara.
5. Michael O. Neary, Alan Phipps, Steven Richman and Peter Cappello, 2000: Javelin 2.0: Java-based parallel computing on the Internet, University of California, Santa Barbara
6. L. F. Lau, A. L. Ananda, G. Tan, W. F. Wong, 2000: JAVM: Internet-based Parallel Computing Using Java, School of Computing, National University of Singapore.