# Design and Implementation of the Software Framework for Distributed Computing

Kin-Yeung WONG, Yin-Man CHOI, and Seng-Wa LAM
Computer Studies Program
Macau Polytechnic Institute
Av. de Luis Gonzaga Gomes
Macao

*Abstract:* - A distributed computing application uses multiple networked computers to work together to accomplish a big task. It can be used to solve calculation-intensive problems such as weather forecasting and astronomical analyzing. There are many common tasks among different kinds of applications. To reduce the development cycle, the goal of this paper is to design and implement an API for constructing distributed applications. In this paper, the core functions of the API are discussed, and a real application written by the API is demonstrated.

*Key-words:* - Distributed Computing, Software Framework, Distributed System, Software Design, Parallel Processing.

## 1 Introduction

Complex scientific problems such as weather forecasting; molecular modelling, air quality simulations, astronomical analyzing, and quantum chemistry are highly calculation-intensive. Supercomputers can be used to tackle the problems. However, not many research parties and organizations cannot afford the deployment and maintenance of supercomputers that are both expensive and space consuming.

Distributed computing environment [1] provides an alternative to supercomputers to carry out the processing-intensive problems due to its flexibility and scalability. A distributed computing application uses two or more networked computers to work together to accomplish a common objective or task. For example, SETI@Home [2], a well-known distributed computing project, makes use of the idle time of millions of desktops in the world, during the screensaver time, to analyze astronomical data to find intelligent life in the universe. Another example is [3] which uses distributed computing to solve Protein problem. Increasing desktop processing power and communications bandwidth makes distributed computing more practical.

The construction of a distributed computing application involves a number of tasks such as task segmentation, task selection and client selection, which are common to most applications. Therefore, an Application Programming Interface (API) for developers to handle the common tasks is desired. The use of API not only effectively shortens the development cycle, but also allows developers to focus on their own project specified functions with less care and concern on the common tasks.

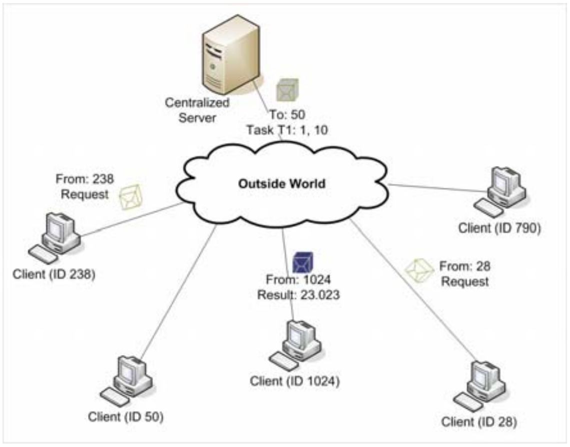

Fig 1. The general structure of centralized distributed computing applications

The goal of this paper is to design and implement of the API for building distributed computing applications. This paper is organized as follows. Section 2 describes about the design of software framework. Section 3 discusses about the implementation of the API and how it achieves the design goals. Then, Section 4 shows the demonstration how the API is used to build a simple distributed application. This API is available for public download at our website [4].

## 2 System Design
Figure 1 shows the general architecture of distributed computing environment, in which there are software agents installed on client nodes and at least one dedicated management server. When a client node is ready to accept job, it will notify the management server and ask for a task. Having finished the downloaded task, the client node will return the result to the server. The task should be done when the client node is idle, and the task can be run in the form of a screensaver or a background daemon. During the execution of the task, if the user need to use its computer, processing of the task will be immediately terminated and return the control to the user.

The management server performs the role of a coordinator. It divides a big job into many small tasks. It also needs to assign those tasks for the available client nodes. Besides, it has to interpret and integrate the results return from client nodes into a meaningful final conclusion.

### 2.1 System Components
Figure 2 shows the proposed software framework for building distributed computing applications. The framework consists of client and server sides. The client side consists of four components, whereas the server side consists of six components. Each component performs a specific task.

Functions for building user interface, boxes in grey, are not provided by the API. It is up to the developers to implementation the interface. The interface can be in a command line mode, or in graphical mode or even in the form of screensaver.

A server side application includes the following components:

| | |
|---|---|
| **Task Allocator** | It is to divide a big job into a number of small tasks for clients. |
| **Task Table** | It is a kind of data structure storing tasks for the clients to download. It also stores the corresponding returned results returned for the tasks |
| **Task Selector** | It selects a job form the task pool, as stored into Task Table. |
| **Server Communication Unit** | It manages the connections between server and clients. |
| **Task Assembler** | When all tasks are finished, this component assembles all the results are stored in the Task Table and form the final outcome. |

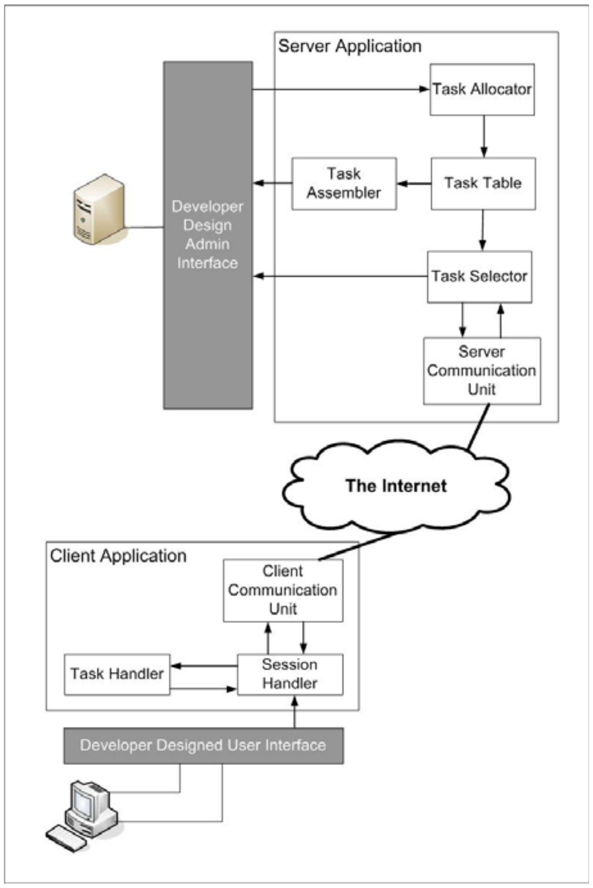

Fig 2. Basic system components of the distributed computing software framework

A client application includes the following components:

| | |
|---|---|
| **Client Communication Unit** | It manages the connections between server and clients. |
| **Session Holder** | It maintains the status of the Task Handler. When the client is available, it asks for a new task from the server. |
| **Task Handler** | It is the component which actually performs the task. After the task is finished, it returns the result to Session Holder. |

## 2.2 Design Criteria of the API

The primary goal of API is to provide a shorter development cycle for developer to build distributed computing applications. Therefore, the design criteria of the framework should include:

| | |
|---|---|
| **Simplicity** | The steps to build an application using the API should be minimized. |
| **Easy to use** | The function calls should be straightforward and involve less number of arguments as well as parameters. |
| **Flexibility** | It should support development of various kinds of application. |
| **Details hiding** | The function should be regarded as a black box and the lower-layer details should be hidden from the developers. |

## 3 Implementation

We implement the software framework of distributed computing (i.e., the components discussed in section 2) using Java language and produce a set of API. In the API, for simplicity, all server components are included in the DistributedServerService class, and all client components in the DistributedRemoteClient class. This section discusses the core classes provided by the proposed API.

### 3.1 Core Classes Provided by the API

#### 3.1.1 Task

The distributed application divides a big job into many small tasks. The Task class represents each task. It includes three important methods:

doTask()

This method uses developers to implement the act work to be done in the client nodes.

setResult(Task t)

To input the result of the work done to Task object. With the consideration of flexibility, the data type of the input variable is the general Object class.

getResult()

To get the result from the t object specified in the setResult(Result t). Developers can cast the general Object to other specific type they want.

#### 3.1.2 DistributedServerService

This class performs functions of the server-side components shown in Fig. 2. The followings are the essential methods:

DistributedServerService DistributedServerService (int port)

This constructor is for developers to initiate the server service object. The port number has to be input into this object in order to form a contactable socket to communicate with clients.

setTaskTableSize(int size)

This method is used to set the size of task table. Note that the input variable, the table size, should be same as the number of sub tasks formed.

activeServer
(TaskAllocator table, TaskAssembler task)

It parses the input objects, table and task, into the object DistributedServerService object and starts up the server. Two objects: TaskAllocator and TaskAssembler are required to be concreted and initiated in advance. After calling this method, the server will start accepting requests from available client nodes.

#### 3.1.3 DistributedRemoteClient

DistributedRemoteClient involves three concrete classes and an abstract class, for developers to build the program the client node. The abstract method doTask()has to be concreted before the DistributedRemoteClient class can be used.

doTask()

This method triggers when the client application receives a task from the server.

requestTask()

This method sends request to the server to get a task to work.

sendBackResult(Task task)

This method returns the result back to the server.

activeClient()

This method starts up the client, after the object of DistributedRemoteClient is initiated.

## 3.2 Merits of the API

The API meet the design criteria

| | |
|---|---|
| **Simplicity and Easy to use** | The basic construction of a distributed computing application requires only the implementation of a few methods. Besides, the methods call involves minimal parameters passing (about two). Therefore, simplicity and easy to use can be achieved. |
| **Details hiding** | Since the details of server and client components are included in DistributedServerServi-ce and DistributedRemot-eClient, respectively, that is, developers do not need to concern the details of the system components. They just need to create and maintain these two classes. |
| **Flexibility** | Different distributed applications have different requirements and ways of tasks presentations are different. For example, protein folding modeling application involves of many data samplings and simulations where as RSA application requires analyzing heavy loading calculation to compute private key. To provide flexibility, the classes for task segmentation, result assemble and job table are defined as abstract classes. On the other hand, most core parameters are passed in the form of general Object type which allows developer to cast it into the specific data type they want. |

## 4 Demonstrations and Applications

This section describes the steps to develop distributed computing applications using our API. It also shows a real application.

### 4.1 Steps of Building Server-side Programs

#### 4.1.1 Preparation of Task Object

Developers use doTask() to define the actual work to be done in client nodes. Developers also need to store the calculated result in the resultObj internal object which can be set and get by setResult() and getResult() respectively.

```
public class MyTask extends Task {
            :
  public void doTask(){
    // The actual work performed in
    // client nodes

    Object r = result
    this.setResult( r );
  }

  public void setResult(Object result){
     this.resultObj = result;
  }

  public Object getResult(){
     return resultObj;
  }
  :
}
```

#### 4.1.2 Building TaskAllocator

The purpose of TaskAllocator is

It is to prepare to construct a task pool for the server to assign tasks to clients. To achieve that, programmers need to fill the abstract method allocateTask() in the TaskAllocator class. The DistributedServerService will make use the TaskAllocator object to do task scheduling.

```
TaskAllocator allo = new TaskAllocator(){

  public ArrayList allocateTask()
  {
    ArrayList al = new ArrayList();

    // New Task objects here and
    // add them into the al ArrayList
   }
};
```

#### 4.1.3 Building TaskAssembler

The abstract method assemble() from the class TaskAssembler is used for developers to assemble the returned results from client nodes. The returned results can be obtained by calling the getAllResult() method.

```
TaskAssembler assembler=new TaskAssembler(){

  public void assemble()
  {
    ArrayList result =this.getAllResult();

    // Write the algorithm to assemble
    // the returned results here
   }
};
```

### 4.1.4 Initialize DistributedServerService
Having created and initialized Distributed-ServerSevice, the server side application can be started up by the method activeServer().

```
DistributedServerService distServer =
               new DistributedServerService(9999);

distServer.activeServer( allo, assembler );
```

## 4.2 Steps of Building Client-side Programs

### 4.2.1 Extending DistributedRemoteClient
During inherent DistributeRemoteClient class, the location IP and the registered port have to be inserted into a self-defined constructor as initialise variables.

```
public class MyDistClient extends
DistributedRemoteClient {

  public MyDistClient(String ip, int port)
  {
        super(ip, port);  }
  }
}
```

### 4.2.2 Concreting doTask()
It is to concrete the abstract doTask()method in the DistributeRemoteClient class. This can be achieved by calling the doTask() method in the Task class.

```
public Object doTask(Object task)
{
  MyTask t = (MyTask)task;
  t.doTask();
  return t.getResult();
}
```

### 4.2.3 Activate the client-side program
When the object of the class extended from extends DistributeRemoteClient has been initiated, the client application can be started up by the activeClient() method.

```
MyDistClient client =
        new MyDistClient( localhost, 9999) ;
client.activeClient() ;
```

## 4.3 Example: Distributed Summation System
We have implemented an example application to demonstrate the development using the proposed API. In the example, the server has 5 small tasks to do. Each task is to sum two integers. When a client is free, it will ask the server for a task to do. After finishing a task, the client will return the result to the server and ask for another task. Figure 3 shows the system environment.

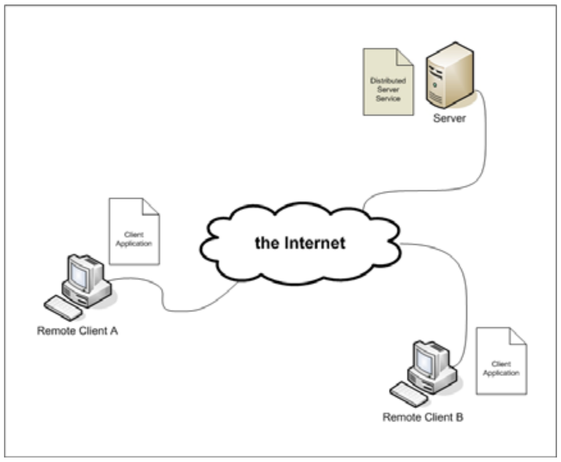

Fig 3. System environment of the example

Figure 4 shows the sample code files for the server-side program. Due to the page length limitation, the code for the client-side program is not shown. This application is run in command line.

As shown in Fig. 4(a), the doTask() method simply adds two integers. The two integers are supplied in the Task object initializations (see the step2 Fig. 4 (b)). When all results are returned from clients, the server simply sums all of them (see the step3 in Fig. 4(b)). Fig. 5 shows the coding of a basic client-side application which just fills the doTask() method, and activate the program.

```
public class MyTask extends Task {
         :
  public void doTask(){

     total = inputA + inputB;

     this.setResult(new Integer(total));
  }

  public void setResult(Object result){
     this.resultObj = result;
  }

  public Object getResult(){
     return resultObj;
  }
  :
}
```
(a) MyTask.java

```
public class MainServer {

  public static void main(String[] args) {

  // 1) create distributed server object
  DistributedServerService distServer = new
  DistributedServerService(9999);

  // 2) build programmer's TaskAllocator
  TaskAllocator alloc = new TaskAllocator(){
     public ArrayList allocateTask(){
       ArrayList al = new ArrayList();
       MyTask t1 = new MyTask("task-A",1,10);
       MyTask t2 = new MyTask("task-B",11,20);
            :
       al.add(t1);
       al.add(t2);
            :
     }
  };

 // 3) build programmer's TaskAssembler
  TaskAssembler asm = new TaskAssembler(){
     public void assemble(){
          int total = 0;
          ArrayList result = this.getAllResult();
          for(int i=0; i<result.size(); i++){
            total=total+
            ((Integer)result.get(i)).intValue();
          }
       }
   };

  // 4) active the distributed server
  distServer.activeServer( alloc, asm );

 }
```
(b) MainServer.java

Fig. 4. Coding for the server-side application.

## 5 Conclusion

Using our API can shorten the development of of distributed computing applications. Although some sophisticated programming library are available, such as BOINC [5] from UC Berkeley, they are usually complex and difficult to work with. To build small-scale distributed computing applications, our API is more appropriated because of its simplicity and flexibility. The proposed API can be publicly downloaded at [4].

*Acknowledgement:*
The work described in this paper is supported by Macao Polytechnic Institute Research Grant (Project No. RP/ESAP-7/2006).

*References:*

[1] M. Lathia, “A useful resource for parallel and distributed computing,” *IEEE Distributed Systems Online*, vol. 6, iss. 4, April 2005.

[2] E. Korpela et al., “SETI@home-massively distributed computing for SETI,” *IEEE Computing In Science & Engineering*, January/February 2001, pp. 78-83.

[3] K. Schreiner, “Distributed Projects Tackle Protein Mystery,” *IEEE Computing In Science & Engineering, January/February 2001*, pp. 13-16.

[4] The API presented in this paper can be downloaded at http://staff.ipm.edu.mo/~kywong/ distributedcomputing/.

[5] Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu/