Dec 2010
Agenda

- Problem Definition & Solution
- Why bothering?
- Implementation Techniques
- Problems (& Solutions)
- HLD (High-Level Design)
- Main Communication Data-Flows
- UI snapshots
- Demo Example: Distributed webpage downloading and parsing
- Performance Analysis
Problem Definition

Main Problem:

*Executing large set of small computational tasks consumes numerous processing time on a single machine.*

- Tasks are homogeneous and non-related
- Executing is serial.
- Execution order is not significant.
Solution

Distribute tasks over more than one machine and exploit computational power of remote machines.
Why bothering?

- Several solution already exist, such as:
  - **Condor**
    - Complex Syntax
    - One task per run
    - Not developer-friendly
  - **MPI**
    - Networking understanding needed
    - Executing and Synchronizing tasks is the user responsibility
  - And more…
Why bothering? (cont.)

- Implementing new solution:
  - User-friendly API. (ease of usage)
  - User transparent.
  - Dynamic System-Management.
  - Task generic.
  - Easy to convert from serial to parallel.
Implementation Techniques

- **Java:**
  - Object oriented
  - Cross-Platform
- **Java RMI (Remote Method Invocation):**
  - Easy to use
  - Transparent networking mechanism
Problems

- Firewall
- Load Balancing
- Auto-Update
- Efficiency
- Execution Transparency
- Fault Tolerance
Firewall

- **Problem:**
  - A firewall can exist between user machine and remote-machines (Executers).

- **Solution:**
  - One side connection (user side)
Firewall (cont.)
Load Balancing (Scheduling)

- **Problem:**
  - Tasks can be submitted to the system asynchronously. Load balancing needed for efficiency.

- **Solution:**
  - Distributing tasks considering remote-machine load, task weight and priority.
  - Round-Robin based.
  - Prevent starvation.
Auto-Update

Problem:

- In regular system, when end-user need to update the tasks and the code executing them (for fixing bugs or changing tasks executing purposes), he needs to go over all remote-machines (Executers) -which can be far far away- and update the code on the machines themselves.

Solution:

- Support the ability of updating the code from the System Manager machine -which is always near the end-user.
  - RMI connection problem arises.
Efficiency

Problem:
- Executing large amount of small tasks one by one could be inefficient (overhead expenses).
- For each task:
  - Scheduling
  - Sending RMI messages

Solution:
- formation of smaller units of information into large coordinated units – Chunking.
- Send a collection of tasks to the same remote-machine (Executer).
Execution Transparency

● **Problem:**
  ● Same I/O operation’s output must be the same in both serial and distributed system.
  ● Feel like running locally.
    ● Output Stream
    ● Exceptions
    ● More…

● **Solution:**
  ● Simulating these operations.
Fault Tolerance

- **Problem:**
  - Remote-Machines may disconnect the network anytime.
  - Executing tasks on the machine will be lost.

- **Solution:**
  - Failure Detector
  - Save the machine state until it connect again, then resumes its work.
High-Level Design
High-Level Design (cont.)

• Concrete Components (Provided by user)
Common Components

- **Item**
  - Base communication item for the System.
    - Task, Result, Chunk etc...

- **Task**
  - Type of task user wants to execute.

- **Result**
  - the result of the Task execution.

- **Chunk**
  - holds a bundle of Items for network optimizations
    - Used for efficiency
Common Components (cont.)

- **Synchronized Counter Mechanism**
  - Used to determine whether the user is idle or not.

- **Log Tracing**
  - Log tracer, resided on each remote-object.
    - Used basically for debug reasons and I/O redirection.

- **Networking Layer**
  - Responsible for communication purposes.
    - Taking in account Firewall existence.
**Executer Component**

- **Main Functionality**
  - Resides on remote-machine waiting for tasks to execute.

- **Task Executer**
  - Holds the concrete task Executer provided by the user.

- **Results Organization**
  - Preparing results for Clients
Client

- **Main Functionality**
  - Resides on user-machine.
  - Provides the implementation for the user API.

- **Results Collector**
  - Polling prepared results from Executers.
System Manager

- Main functionality
  - Match-making between *Clients* and *Executers*. (Scheduling)
- Holds lists of *Executers/Clients* connected
- Manages common system operations
  - Auto-Update
  - Clean Exit
  - Etc...
- Failure Detection Mechanism
Client System Data-Flow
Executor System Data-Flow
Main User Scenario

- Task1
- Task2
- Client
- System Manager
- Executer 1
- Executer 2
- Executer 3
- Result1
- Result2
- Updates
- Firewall
User Interface
User Interface

- Executers Tab
User Interface

- Executer Trace Log
User Interface

- Update Tab
Demo Example:
Distributed webpage downloading and parsing

- Distributed webpage downloading and parsing demo
- Created in order to test
  - System performance
  - Ease of usage
- From Serial code to Distributed using the system API
- Tested on Windows (XP) and Linux (Suse)
Before

```java
public class DownloadFiles {
    private final String rootDir_
    private final DocIndexRepository downloadedDocs_
    private void run(String inputFileName) throws Exception {
        BufferedReader input = FileHandler.openFileReader(inputFileName);
        while (true) {
            String url = input.readLine();
            try {
                String text = downloadAndParseFile(url);
                String fullFileName = createOutputFile(rootDir_);
                writeResultToFile(text, fullFileName);
                outputFileStream.close();
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

This **Line** needs to be distributed to make it possible run over more than one machine.
Changes to be done

- Implement
  - Task class
  - Result class
  - Executer class (downloadAndParseFile function)

- Modify the code using the API
**Task Code:**

```java
import diSys.Common.Item;

public class DownloadTask extends Item {
    public String url;
    public DownloadTask() {
        super(0);
        this.url="";
    }
    public DownloadTask(long id, String url) {
        super(id);
        this.url=url;
    }
    public String toString(){
        return "Task ID: " + this.getId();
    }
}
```

**Result Code:**

```java
import diSys.Common.Item;

public class DownloadResult extends Item {
    public String text;
    public String url;
    public DownloadResult(long id) {
        super(id);
    }
}
```
Executor Code:

```java
public class DownloadExecuter implements IExecutor<DownloadTask, DownloadResult> {

    public DownloadResult run(DownloadTask task) throws Exception {
        DownloadTask task = task;
        DownloadResult res = new DownloadResult();

        // Output will be redirected (appears on Client machine)
        System.out.println("trying to download url: "+task.url);

        // Do the job...
        res.text = downloadAndParseFile(task.url);
        res.url = task.url;

        System.out.println("url: "+task.url + " downloaded!");
        return res;
    }
}
```
public class DownloadClient {

private final String rootDir_;  
private final DocIndexRepository downloadedDocs_;  

public static void main(String[] args) throws Exception {
    //Initialization (connecting to system manager)
    RemoteClient<DownloadTask, DownloadResult> client =
    new RemoteClient<DownloadTask, DownloadResult>("localhost", 5000 /*port*/, 10 /*chunk size*/);
    //Start the Client  
    client.Start();
    BufferedReader input = FileHandler.openFileReader(inputFileName);

    while((String url = input.readLine())!=null) {  
        client.addTask(new DownloadTask(url)); //Submit tasks to System  
        //String text = downloadAndParseFile(url);  

    }

    while(client.GetTaskNum() > 0) //While there are tasks in execution, try to get results  
    try {
        DownloadResult dr = client.GetResult();  
        //May throw exception (if exception thrown in Executer code)
        String fullFileName = createOutputFile(rootDir_);
        writeResultToFile(dr.text, fullFileName);
        outputFileStream.close();
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
Performance Analysis

Study on a sample application: Distributed webpage downloading and parsing

- The system has been tested with the following workload: the Task is downloading a web page given its URL, and the Result is text extracted from the HTML of the web-page using external HtmlParser library.
Benchmark #1

- the Performance (Total time) of executing 150 Tasks on 1, 2, 4 and 8 Executers which ran over 2 windows different machines

Distributed System Benchmark #1
Benchmark #2

- the Performance (Total time) of executing 150 Tasks on 3, 6, 9 and 12 Executers which ran over 3 machines
- (2 windows machines and 1 Linux machine).

Distributed System Benchmark #2

![Graph showing the relationship between the number of executers and the time (seconds). The graph indicates a decrease in time as the number of executers increases.]

- Time (sec)
- Number of executers
- Num of Tasks: 150
- Num of Machines: 3