An Simulation of MapReduce as a Higher-Order Web Service in Cloud

Desheng Li        Chuanchang Liu         Na Deng        Changbao Li        Junliang Chen
State Key laboratory of Networking and Switching Technology
Beijing University of Posts & Telecommunications
Beijing, China
e-mail: ldsyy2006@126.com; lcc3265@bupt.edu.cn; iamdengna@gmail.com; lichangbao_1@163.com; chjl@bupt.edu.cn

Abstract—As the emergency of considerable cloud platform, researchers and developers are turning their attentions to the architecture of cloud service environment. This paper proposed a concept of high order web service supporting MapReduce programming Paradigm and related cloud service platform architecture. Moreover, to solve the embarrassment of complicated configurations and testing, a cloud service simulation platform is proposed to verify the feasibility of the architecture. Finally, a typical example of MapReduce is simulated on that system to illustrate the correctness and performance.

Keywords—cloud; web service; MapReduce; high order

I. INTRODUCTION

Cloud computing is an emerging technology that allows the users to conveniently access and share kinds of resources – processes, data and networks, which are distributed over the Internet. A major challenge in cloud computing is to import concept of service, such as IaaS(Infrastructure as a Service), PaaS(Platform as a Service), and SaaS(Software as a Service).

Mapreduce[1], which stems from the ideas of functional programming, is an important infrastructure in cloud which takes into account the fault tolerance and simplicity to use initially is attracting a lot of attentions. As the emergency of Google’s Mapreduce framework[1] and its open-source implementations such as Hadoop[2], researchers and developers begin to build their own platform to do some large-scale computing for the distributed data.

Although the concept of component has been proposed for a long time, but how to implement the equivalence of high order function in traditional programming language is not realized well in SOC. In literature[3], a kind of component, Higher-Order Components(HOCs) is proposed, which can be parameterized not only with data but also with application-specific code. Moreover, it cab be used to implement various patterns of parallel and distributed processing, such as farm, pipeline, divide-and-conquer and others.

Based on the above work, we proposed a novel kind of web service, called Higher-Order Web Services(HOWSs), to implement the MapReduce paradigms in cloud service environments. In order to test the algorithms in MapReduce, we also design a platform to simulate the cloud service environment.

II. HIGHER ORDER WEB SERVICE AND MAPREDUCE

A. MapReduce

In programmer’s perspective, the operators in MapReduce are inquired to complete several stub functions, including a map function, a partition function, a combine function, a reduce function, and merge functions. Here, the combine function is just treated as the local version of reduce one. The map functions take input records and generate intermediate key-value pairs. The partition function given the key and the number of reducers can provide the index of reduce where the output of all of the maps is allocated. The reduce function takes all the values for an intermediary key and iterate through the values that are associated with that key and output the result of processing these values. And the merge functions could merge the results of reduce and other data into a new one, find the differences of them, and do something other stuff. These functions are easy to write, more important, application-specific.

B. Higher Order Web Service(HOWS)

Higher-Order Web Services (HOWSs) are novel components in SOC that can be widely used in various cloud service environments. As they are general generic software components, so they can be parameterized and customized well. Common components can only be customized with data, but HOWSs can be parameterized with both data and runtime codes. Different with the definition of HOCs in [3], we don’t specify the patterns of parallel, such as farm, pipeline, divide-and-conquer. Instead of it, these patterns are chosen by cloud service platform automatically.

Each HOWS consists of two parts: One is a collection of interfaces describing the signatures of parameters in HOWS; the other is a server-side implementation. The interfaces of HOWS is more friendly than the traditional higher order functions. To comply with the corresponding standards in web...
service field, we give the definitions in the form of WSDL as shown in Fig. 1 without some trivial details.

```xml
<portType name="sHowS">
  <operation name="setConf"/> ...
  <operation name="setHighOrderOpt"/> ...
  ...
  <operation name="run"/> ...
  <operation name="getRunningStatus"/> ...
  <operation name="getResultResponse"/> ...
</portType>
```

Figure 1. Interface Definition of a HOWS in WSDL.

One advantage of using HOWSs is that the applicational programmers need not to consider the multifarious details of various cloud environments and the communication of different components. Moreover, HOWSs can not only be parameterized, but also composed as the common web service component. But this kind of composition is more difficult since the parameters of component consist of both data and codes. In this paper, we don’t discuss the composition of HOWSs.

C. MapReduce as a HOWS

In our research, we consider MapReduce as a kind of HOWSs, called MapReduce HOWSs. Take the implementation in Hadoop[2] as the example, an instance of a MapReduce HOWS can be seen as a job instance, which specifies the parameters of the job, such as input/output paths, the number of Map and/or Reduce tasks and so on. In addition, we can run an instance of that job, return the status of runtime and get the result messages by a MapReduce HOWS. A basic interface of a MapReduce HOWS is shown in Fig. 2. From that, we can get the subclasses of the MapReduce HOWS, those are MapPT and ReducePT in Fig. 3.

```xml
<portType name="sMapReduceHowS">
  <operation name="setConf"/> ...
  <operation name="setReducer"/> ...
  <operation name="getRunningStatus"/> ...
  <operation name="getResultResponse"/> ...
</portType>
```

Figure 2. Interface Definition of a MapReduce HOWS in WSDL.

```xml
<portType name="MapPT" xmlns:tns="howsisse:HW">
  <input name="mapperInfoInput" messages="tns:MapperInputMsg"/>
  <output name="mapperInfoOutput" messages="tns:MapperOutputMsg"/>
  <fault name="mapperFault" messages="howsisse:MapperFaultMsg"/>
</operation>
</portType>
```

```xml
<portType name="ReducePT" xmlns:tns="howsisse:HW">
  <input name="reducerInfoInput" messages="tns:ReducerInputMsg"/>
  <output name="reducerInfoOutput" messages="tns:ReducerOutputMsg"/>
  <fault name="reducerFault" messages="howsisse:ReducerFaultMsg"/>
</operation>
</portType>
```

Figure 3. Interface Definition of a Map and Reduce in WSDL.

III. DESIGN AND IMPLEMENTATION OF THE PLATFORM

A. Architecture of Higher-Order Cloud Service Platform

To compile, test and run a MapReduce program are a complex and error-prone things by far. And most of times these need the supports of related cloud environments. But to most researchers and developers, it is slightly expensive to use cloud environments. So in this sense, to design and implement a simulation platform of cloud service is important to support MapReduce paradigm. The architecture of the platform is given in Fig. 4.

![Architecture of MapReduce Simulation Environment](image)

From Fig. 4, we can see that clients call the Higher-Order Web Service by Web service and/or manual requests via internet. The Higher-Order Web Service offers a well-defined paradigm of parallel process. Different with traditional web service, Higher-Order Web Service should be given some parameters, which include both data and codes. Data parameters stand for the resources identifier of dataset, herein is URL; while code parameters specify an executable code, such as Map, Reduce and Merge Java byte codes. The codes can also be transmitted by the designers via internet, stored in the code base, and checked by code key. Note that the code base does not need to be rational data base. We take the key-value database as the implement. When the MapReduce Portal service is told to run a job, the MapReduce Executing Engine run the task with the parameters, code and dataset. When the task completed or failed, the MapReduce Portal service could also response to the client.

B. Mobility of Code

As our purpose is to offer a platform to simulate the algorithms of MapReduce, so the executing engine is only run on a single machine. Thus the mobility of code is not very complex. We take the reflection of byte code in Java as the main technique. Although there exist some manners to get the code:

1. Transmitted as byte array in SOAP message;
2. From code base by key;
3. Represented as a form of XML, like BPEL as in literature[4].

In our implement, we use (2) to get the code. After the acquirement of code location, we reflect the byte code by a
class called coderClassLoader shown in Fig. 5, and instantiate
the objects of code.

```java
public void setMapper(Mapper mapper) throws InstantiationException, IllegalAccessException {  
  //
  Class mapperClass = coderClassLoader.loadMapper();
  this.mapper = (Mapper)mapperClass.newInstance();
}
```

Figure 5. Codes of Reflection.

C. Design of Concurrent Threads Lib

To simulate the nodes in distributed environment, we
import a concept of task on thread. Each node can be
represented as a task on thread, but not a thread itself. But the
simulation of data storage of node is slight inconvenient. We
use the external file with the key of task and textual data
structure. To implement our concurrent task on thread package,
we take the Future model [5] as the main approach, which has
been applied widely in the world of parallel and concurrence.
Because of its trait of asynchronism, the object referred by it
may not available now. When the flow reaches the object, there
exist two choices: one is expect object to run and doing the
sequent operations; the other is when it is still unavailable, the
flow enter the some alternative choice.

In our system, we use java.util.concurrent, FutureTalker[6]
to interact with the caller of thread. As Future<V> being the
interface of return a result of a asynchronic task, FutureTalker
is an well implemented instance of it. Moreover, it has the
ability of offering listener, which can be informed when the
task completed or failed. The advantage of it is that there is no
need to roll robin the executing status of tasks.

IV. EXPERIMENT OF SIMULATION

A. Evaluation of Pi with Monte-Carlo Approach

The idea of evaluation of Pi with Monte-Carlo approach in
[7][8] is to simulate plenty of random drops in a square and its
inner quarter sector. As in Eq. (1) shown, the ratio of number of
drops in sector n_2 and the total number of drops n_1 pro rata
with their areas.

$$\pi = \frac{4n_2}{n_1}$$  (1)

As the operations with drops including generating,
judgment, and summation don’t need the communication
between processes, so it is natural to be implemented by
MapReduce with Share-Nothing idea. The Map and Reduce
procedures are given in Fig. 7 and Fig. 8, which follows the reference[9].

![Map Function of Monte-Carlo Pi.](image)

![Reduce Function of Monte-Carlo Pi.](image)
B. Experiment and Result

In order to testing the simulation platform with the evaluation of Pi, we design this experiment. The precision of the evaluation is determined by the total number of the drops, i.e., \( n = mb \). Where \( m \) is the number of Map and \( r \) is the number of Reduce, while \( b \) the repetition factor of each Map. The configure of parameter and result of experiment is given in Table I. Herein, the value of \( b \) is fixed to 1000 and \( r \) to 2.

<table>
<thead>
<tr>
<th>( m )</th>
<th>( \text{Pi} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.176</td>
</tr>
<tr>
<td>2</td>
<td>3.106</td>
</tr>
<tr>
<td>4</td>
<td>3.182</td>
</tr>
<tr>
<td>8</td>
<td>3.144</td>
</tr>
<tr>
<td>16</td>
<td>3.136</td>
</tr>
<tr>
<td>32</td>
<td>3.141875</td>
</tr>
<tr>
<td>64</td>
<td>3.1456875</td>
</tr>
<tr>
<td>128</td>
<td>3.14121875</td>
</tr>
<tr>
<td>256</td>
<td>3.14671875</td>
</tr>
<tr>
<td>512</td>
<td>3.142515625</td>
</tr>
<tr>
<td>1024</td>
<td>3.14609375</td>
</tr>
<tr>
<td>2048</td>
<td>3.1418359375</td>
</tr>
<tr>
<td>4096</td>
<td>3.141623046875</td>
</tr>
<tr>
<td>8192</td>
<td>3.1415156252568</td>
</tr>
<tr>
<td>16384</td>
<td>3.1415947265625</td>
</tr>
<tr>
<td>32768</td>
<td>3.14153564453125</td>
</tr>
<tr>
<td>65536</td>
<td>3.14155895996094</td>
</tr>
<tr>
<td>131072</td>
<td>3.14157601928711</td>
</tr>
<tr>
<td>262144</td>
<td>3.14159896850586</td>
</tr>
</tbody>
</table>

From Table I, we can conclude with that as the number of Map nodes getting larger, the evaluation of Pi become more precise but with more executing time. To our surprise, the maximal valid number of Map thread could reach to a very big number, which is even not inconceivability in a real middle-scale cloud environment.

The executing time of the algorithm is shown in Fig. 9. From it, we can see when the number of Map nodes ranges over 0 to several hundred, the executing time increase not fast because the parallel of Map threads. But when the number reaches one thousand, the executing time increase very fast since the cost of each thread make the system slower than that of fewer nodes.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce the equivalence to high order function in SOC, High Order Web Service (HOWS), with corresponding representation models and architecture. We image the elements of HOWS into a common WSDL file to keep the compatibility of standards. Then a simulation platform is proposed with main techniques of implementation, such as mobility of code and simulation of nodes. The experiment shows that our platform can simulate the MapReduce program well in form of HOWS with more capability of nodes.

In future, we consider importing the logic of control flow into a form of XML. Hence HOWS is also represented in that form. In practice, we plan to explant our platform to multi-nodes cluster, in which a thread can be labeled by thread ID and IP address of hosted node.

ACKNOWLEDGMENT

This work is supported by “973” program of National Basic Research Program of China (No. 2007CB307103), and also supported by the Fundamental Research Funds for the Central Universities (No.2009RC0507).

REFERENCES