International Journal of Grid Distribution Computing
Vol.7, No.4 (2014), pp. 177-186
http://dx.doi.org/10.14257/ijgdc.2014.7.4.17

Cloud Service Platform for Julia Programming on Supercomputer

Zhang Changyou[1], Liu Renfen[2], Duan Shufeng[3], Zhu Xiaomin[4] and Liu Wei[4]

[1]Institute of Software, Chinese Academy of Science, Beijing 100190, China
[2]Sifang institute, Shijiazhuang Tiedao University, Shijiazhuang 051132, China
[3]Shijiazhuang Tiedao University, Shijiazhuang 050043, China
[4]National SuperComputer Center,Jinan 250101, China
E-mail: changyou@iscas.ac.cn

Abstract

It is difficult to deeply develop computing ability of supercomputer through programming. There are two approaches to resolve this problem. The first is that programmer masters advanced knowledge about architecture of the supercomputer. The second is to construct a friendly programming environment to ease the difficulties in parallel programming. Julia is a scripting language which supports language-level high performance computing. It is easier to implement parallel algorithm in Julia language. So, we develop a programming supporting environment for Julia to edit and debug source code, and prepare a set of portable function library for heterogeneous supercomputers. These programs are sent to supercomputers through a message system. As job description, these Julia codes are scheduled to proper nodes and executed based on the portable function library. In the computing case of bus-line statistics, the parallel Julia program on 4 nodes achieved a speed up of 3.15x.

Keywords: *Cloud computing; Julia language; Supercomputer; Message system*

1. Introduction

The architecture of latest supercomputer is more complex. Many supercomputers in Top500-list are of heterogeneous architecture [1]. Such as the Tianhe-2 - MilkyWay-2 (TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P), and the Titan - Cray XK7 (Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x). It retained its position as the world's No. 1 system with a performance of 33.86 petaflop/s on the Linpack benchmark.

Programming on heterogeneous supercomputer is very hard. There are two approaches to resolve this problem. The first is that programmer masters advanced knowledge about supercomputer architecture. The second is to construct a friendly programming environment to ease the difficulty of parallel programming.

Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments[2]. It provides a distributed parallel execution, numerical accuracy, and an extensive mathematical function library.

Julia's LLVM-based just-in-time (JIT) compiler combined with the language's design allows it to approach and often matches the performance of C. The core of the Julia implementation is licensed under the MIT license [3]. Users can combine Julia with their own C/Fortran code or proprietary third-party libraries. Furthermore, Julia makes it simple to call external functions in C and Fortran shared libraries, without writing any wrapper code or even recompiling existing code.

We constructed a cloud service platform to make programming easier on supercomputers. This cloud platform paves the way for fully cloud-based operation, including data management, code editing and sharing, execution, debugging, collaboration, analysis, data exploration, and visualization. The eventual goal is to let people stop worrying about administering machines and managing data and get straight to the real problem.

The remainder of this paper is organized as follows: Section 2 introduces the architecture of Julia cloud platform. Section 3 describes the design and implementation of cloud platform. Section 4 illustrates a use case on Beijing bus lines. Section 5 lists the main sub-processes of programming on this cloud platform. Section 6 gives the conclusions and further discussion.

2. Architecture of Julia Cloud Platform

The architecture of Julia cloud platform consists of three components: User Programming Environment (UPE), Message Passing System (MPS), Julia Running Environment (JRE). The architecture of this platform is illustrated in Figure 1.

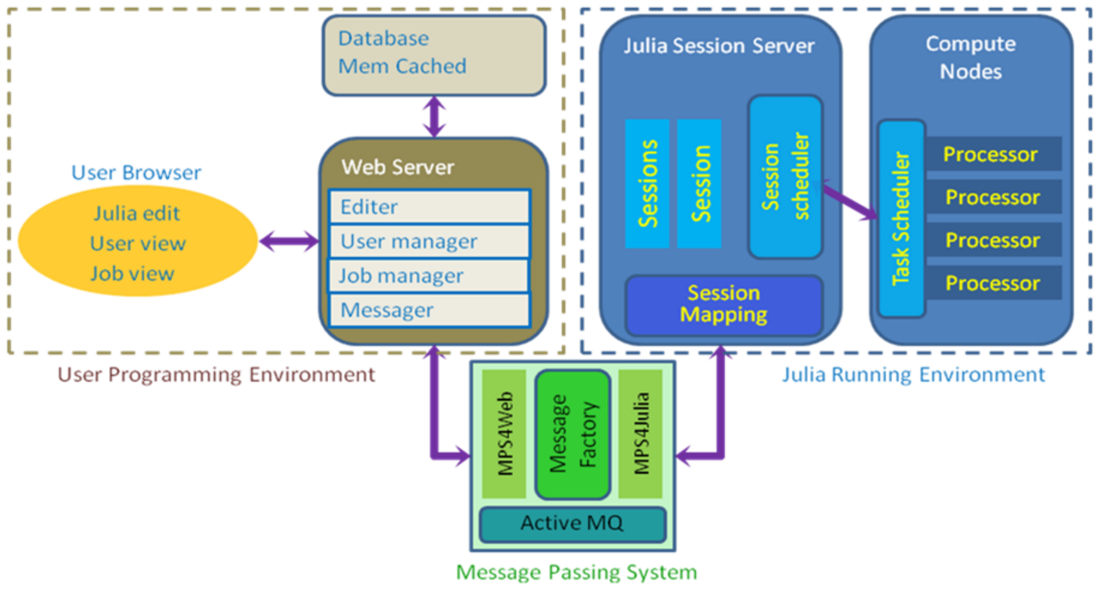


Figure 1. Architecture of Julia Cloud Service Platform

2.1. User Programming Environment (UPE)

UPE is a web-based system with functions such as, Julia code editing, Job information management and User information management. UPE compose a Web server and many User Browsers. Julia Editer is a IDE for Julia coding. Users log on and edit their source codes on browser directly. User information and job information is maintained in database to support user logging and code editing, submitting the finished program, and checking job running.

2.2. Message Passing System (MPS)

MPS is responsible for passing user’s code file and related resources to JRE(Julia Running Environment) and feedback the running status/result information to UPE. MPS is constructed on ActiveMQ server [4]. We develop MPS on the open API.

In this Julia cloud platform, there are five types of messages, which are listed in Table 1.

Table 1. Message Types

Type	Description
JOB_SUB	Job submission message with the content of Julia source file
JOB_QRY	Job querying message which ask system to feedback job's status
FDBK_JRST	Job results message with result file
FDBK_JSTA	Job status message from computing node
FDBK_RINF	Job running information message from computing node

2.3. Julia Running Environment (JRE)

User's job-submitting will create a session with a long life till all associated activities are finished. A job's life often spans a very long period, maybe several days. User will log out when he finished his source code submit. And he will log on system again to check the job running status. During this job running period, system will keep this session bond with the job.

Before the job's running, it waits in a queue till this running condition ready. If there are two or more supercomputers, this session scheduler negotiates with the local task scheduler on supercomputers. Job running information and results are encapsulated as messages back to mapped UPE.

2.4. Supercomputer

Supercomputer is a set of computer connected with networks. Often, it is of large quantity of CPUs and huge memory capacity. All jobs run on the supercomputer with the dispatching of local task scheduler. Supercomputer run Julia program based on its running environment.

3. System Design

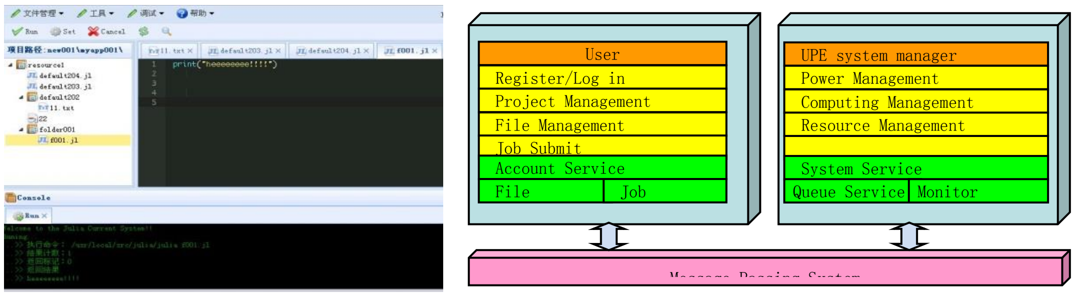
System follows Object-Oriented design policies. Designing scheme is described in UML (Unified Modeling Language)[5].

3.1. User Programming Environment

User edits programs in UPE, which provides a code editing interface and submitting/debugging window to observe running results. The interface includes three parts. The bottom is result-monitor window which displays program running information. Left above is a project structure window which displays the linked files and resources to this project. Right above is the code editing window in which user writes/edits his code file. User submits his project by pressing the Run button or click Run command in menu tools. The layout of the interface and the function stack are illustrated in Figure 2.

3.2 Message Passing System

3.2.1 System Deployment: Messages are passed from Web server to Julia Session Server, and vice versa. There are three levels in MPS. The top level includes two parties of messenger, Web Server and Julia Session Server. The second level is interfaces of message sender and receiver. The bottom level is Active MQ Server, which is an open source message queue server developed by Apache. Active MQ Server level provides connection. Active MQ Server supports multi-thread concurrently sending and asynchronous multi-thread receiving.



```
print("heeeeeeee!!!!")
```

```
Welcome to the Julia Current System!!
Running
>> 执行命令： /usr/local/src/julia/julia f001.jl
>> 结果计数：1
>> 返回标记：0
>> 返回结果
>> heeeeeeee!!!!
```

Figure 2. Layout of the Interface and the Function Stack of UPE

3.2.2 Message Format: To satisfy the requirements of top applications, a message is formatted into three parts. The first part is Message Head which contains of attributes such as, Message ID, Sender, Receiver, Message Type, Sending Number, Sent-time, Version, Priority, Length of body. The second part is Message Body which contains Message Content (Session ID, Resource Description, Message Text, Source file name, Source file type, File order, File number) and Message Extension (Extent Content, Reserved field). The third part is Message Tail which contains Message Digest Algorithm, Message Digest, Checking Option.

The format of message is shown in Figure 3.

Message	Head		message ID
			sender（IP+Port）
			receiver（IP+Port）
			message type
			sending number
			sent-time
			version
			priority
			length of message body
	Body	message content	session ID
			resource description
			message text
			source file name
			source file type
			file order
			file number
		extension	extent content
			reserved field
	Tail		message digest algorithm,
			message digest
			checking option

Figure 3. Format of Message

3.2.3 Major Classes Design: There are 4 Java packages in MPS, *configure*, *view*, *message*, *control*. Package *message* is the base to construct message passing system. Package *message* contains three entity classes. They are MsgHead, MsgContent, MsgTail. We omit the details of methods to consize the UML diagram of package *message* as Figure 4 And the UML diagram of package *configure* is illustrated in Figure 5.

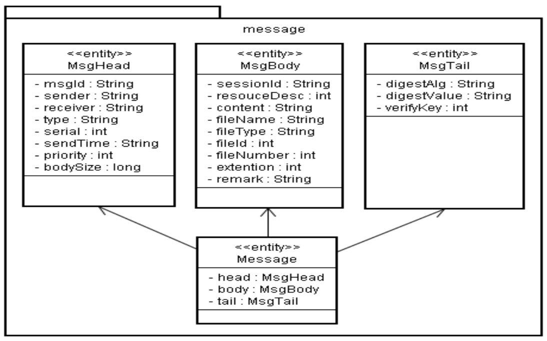


Figure 4. Structure of package of message

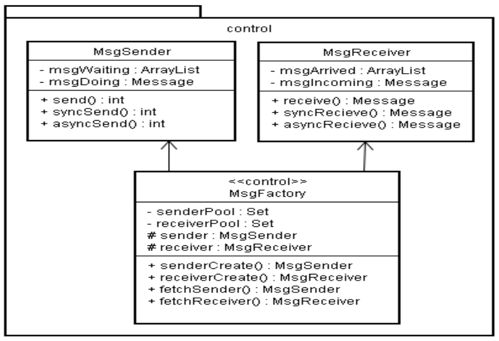


Figure 5. Structure of package of control

There are three classes in package control (MsgSender, MsgReceiver, MsgFactory). MsgSender is responsible for sending message and it is connected to the sending interface of Active MQ server. MsgReceiver is responsible for receiving message and it is connected to the receiving interface of Active MQ server. MsgFactory is responsible for the control processes of message. It manages the muti-threads creating/running to ensure sending/receiving/buffering safe and reliable.

Both class MsgSender and MsgReceiver extend class Thread, support 3 model (default, asynchronous, synchronous). MsgFactory is responsible for creating MsgSender and MsgReceiver.

3.3. Julia Running Environment

Rear to the message passing system, there is a session scheduling system which dispatches jobs sessions to the local task scheduling systems on supercomputer.

3.3.1 System Deployment: Our session scheduling system plays a role of session interface to supercomputer. The deployment of this interface is illustrated in Figure 6.

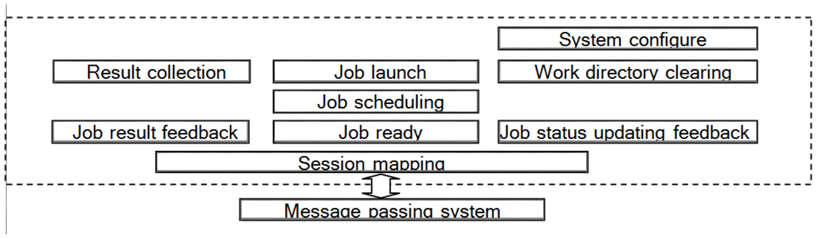


Figure 6. Deployment of Session Scheduling System

3.3.2. Major Classes Design: There are three packages in JRE. They are *sessioninterface*, *jobmanager* and *configure*. Package sessioninterface includes class CloudSession and SessionMapping. Package jobmanagement is the biggest one. It is responsible for job starting and observing on supercomputer. Package jobmanager includes class JobPreparer, JobStatManager, JobResultManager, JobRunningCondition, Job, JobRunner, JobSchedler. Package configure includes classs JobWorkDirInfo, WorkDirManager and SystemConfigure. Class WorkDirManager is responsible for creating and cleaning work directories. The structure of package jobmanagement and configure is illustrated in Figure 7.

Figure 7. Structure of Package Job Manager and Configure

4. Use Case of Bus-line Statistics in Beijing

4.1. Use Case Description

In Beijing, there are approximately 3723 bus lines totally [6]. Each bus line consists of 40 stops in maximum. And a bus is launched every 5 minutes (about totally 200 bus launched). Based on bus running data in one day, these data will be modeled in a 3-dimmension array. Each element in this model is the time when bus reached this stop. Finally, the problem is to calculate the average running hour between every two bus-stops for each bus line. This interval statistic is important to transportation optimization [7].

Julia supports multi-dimension array in language level. The type of elements is Int32. The computing results are put in a 2-dimension array with element-type of Float32. The data model of this computing case is illustrated in Figure 8.

In Figure 8, the axis X denotes bus-line identifier. The axis Y denotes the stops in each line. The axis Z denotes the launching order for each bus.

4.2. Computing Model

In Julia programming, the CPU and kernel of CPU are called as processor. We give 4 processors in this computing case. The steps of parallel computing in Julia are illustrated in Figure 9.

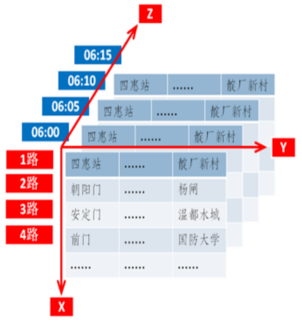


Figure 8. Data Model of Bus Line in Beijing

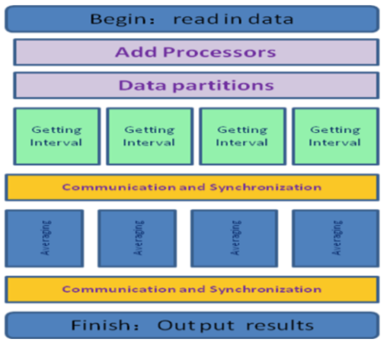


Figure 9. Steps of Parallel Computing in Julia

Comparing with the serial program, the parallel program in Figure 9 calls function addprocs() to prepare other 3 processors computing together. The total process of computing consists of twice computing and twice synchronizations.

4.3. Computing Model

The source file of main program of this computing case is listed in Figure 10. Begin of this program, the problem scale (const X=4000; const Y=40; const Z=200;) and the processor number (const NP=4) are defined. The calculating of bus running period and averages are defined in separated files as functions [8].

```
const X=4000; const Y=40; const Z=200;const NP=4;
XX=4000/NP; busIni = Array(Int32,X,Y,Z)
... ...
busInter = Array(Int32,X,Y,Z)
addprocs (NP-1)
... ...
pbInter1=busInter[1:XX,1:Y,1:Z]
pbInter2=busInter[XX+1:2XX,1:Y,1:Z]
pbInter3=busInter[2XX+1:3XX,1:Y,1:Z]
pbInter4=busInter[3XX+1:X,1:Y,1:Z]
require("zm.jl")
rzm1=remote_call(1,zm,pbInter1,XX,Y,Z)
rzm2=remote_call(2,zm,pbInter2,XX,Y,Z)
rzm3=remote_call(3,zm,pbInter3,XX,Y,Z)
rzm4=remote_call(4,zm,pbInter4,XX,Y,Z)
pbInter=vcat(fetch(rzm1),fetch(rzm2),fetch(rzm3),fetch(rzm4))
```

Figure 10. Julia Source File of Computing Case in Data Parallel Model

4.4. Experimental Results

We do these experiments on a cluster with 4 processors. The operating system is Ubuntu12.04, and the version of Julia is Beta 0.2.0. The elapsed time of serial program and parallel program are listed in Table 2.

Table 2. Elapsed Time of Serial and Parallel Program

Algorithm	Adding processors	Data partition	Calculate period	Averaging	Total（s）	Speed up
Serial	--	--	11.67	11.81	23.58	3.15x
parallel	2.70	0.34	2.02	2.43	7.49	

The experimental results in table 2 show that the calculate-period shortens from 11.67s to 2.02s. And the step of adding-processors/data-partition cost extra 2.70s/0.34s. The total elapsed time shortens from 23.58s to 7.49s. The speed up reaches to 3.15x.

5. Process of Cloud Computing

From the above design, we get how a Julia program is created and run on a supercomputer, and how a user gets his calculating results. In summary, the main event-stream is listed as follows.

a. register to be a user of cloud service platform
b. create a project
c. edit Julia source file with requisite resources such as .dll file and so on.

d. command to run or debug program
e. save source files and linked resources and wrap them to one file
f. create a session
g. create a job message consists of wrapped file and session ID
h. job message is sent to Active MQ server and queued
i. session schedule service fetch this job message
j. session schedule service unwraps Julia source file and linked resources
k. create work directory and save Julia source file and linked resources
l. create job description for job scheduler on supercomputer
m. job status update locally, and send message to user environment via message system
n. run job on supercomputer and get results
o. notifying-message is created and sent to user environment via message system
p. update job status in user environment
q. user check his job status in database and know his job finished
r. user check program running result
s. if result satisfies user expectations, notify running environment session finished
t. running environment mark the work directory is clearable
u. clear directories in clearable list
v. work on a new project or develop it
w. user log off

In fact, there are many other steps to handle the exceptional event stream. These details are merely keeps in the design files.

6. Conclusion and Further Research

To help programmers take advantage of supercomputers, we constructed a cloud platform for Julia programming. The User Programming Environment is a user-centered programming interface, which supports projects management and job status management. The Julia Running Environment sockets to the supercomputer. It is responsible for dispatching job-sessions onto supercomputers' local task scheduler and for monitoring/reporting the status of job running. These two end environments negotiate via a Message Passing System. Message encapsulates all the information for job running and debugging. We designed this platform in UML and implemented it in Java language based on some open source projects, such as active MQ server.

Julia language is a developing language which is very suitable for implementing some parallel algorithms with performance closed to C/Fortran language. We calculated bus-lines of Beijing in Julia language. And we gained speed up in 3.15x. Program refining was doing and accumulated some experiences in performance improving.

For heterogeneous super-computers, the parallel computing abilities need to be extended to various architectures, for example GPU (Graphics Processing Units), MIC (Many Integrated Core), and so on. In this case, we prepared a set of dynamic link library coded with C/Fortran/CUDA, and took the advantages of the abilities of calling external functions in C and Fortran shared libraries. In future, our main objective is to develop our libraries for a given running platform to support more complex applications [9].

Acknowledgments

This paper is supported by the Natural Science Foundation of China (61379048), Special Project of National CAS Union – The High Performance Cloud Service Platform for Enterprise Creative Computing, Special Project of National CAS Union – The Key Issues of

Intelligent Transportations, the National High-Tech Research and Development Plan of China under Grant No. 2012AA010902-1. The authors would like to thank the anonymous referees for their helpful comments from which the preparation for this version of the paper has benefited.

References

[1] Top500 List - November 2013. http://www.top500.org/list/2013/11/
[2] http://julialang.org/
[3] http://en.wikipedia.org/wiki/MIT_License
[4] http://activemq.apache.org/
[5] http://www.uml.org/
[6] http://bus.17u.com/bus/beijing/
[7] R.-M. Li, J.-Z. Liu, F.-H. Zhu, "On Computational Experiment in Parallel Public Transportation System", ACTA AUTOMATICA SINICA, vol. 39, no. 7, (**2013**) July, pp. 1011-1017.
[8] C. Zhang and X. Zhang, "Julia Language and its Parallel Computing Support", Keynotes Report on 6th R Language, (**2013**) May 18, Renmin University, Beijing, China.
[9] J.-y. Wu, J.-q. Fu, L.-d. Ping, Q. Xie, "Study on the P2P Cloud Storage System [J]", Acta Electronica Sinica, vol. 35,no. 5,(**2011**), pp. 1100-1107.

Authors

Zhang Changyou, he received his Ph.D degree in Beijing Institute of Technology, Beijing, China. He is a professor in Institute of Software, Chinese Academy of Science. His current research interests include parallel programming, computer-supported cooperative work, intelligent information network.

Liu Renfen, she received her master degree in Shijiazhuang Tiedao University, Shijiazhuang, China. She is a lecturer in Sifang institute of Shijiazhuang Tiedao University. She current research interests include network security, P2P, Web mining, parallel programming.

Duan Shufeng, she received her master degree in Shijiazhuang Tiedao University, Shijiazhuang, China. She is a lecturer in School of Information Science and Technology, Shijiazhuang Tiedao University. Her current research interests include image processing, Web applications, cloud computing.

Zhu Xiaomin, he received his Ph.D. degree in Institute of Computing Technology, Chinese Academy of Sciences. He is a assistant professor in National SuperComputer Center in Jinan. His current research interests include parallel computing and high performance computing.

Liu Wei, he received master's degree in xi'an petroleum university, xi'an, China. He is a Research Assistant In national super computer center of jinan. His current research interests include High performance computing, research and development of parallel programs, Web applications, databaseoptimization.