INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING, *SERIES B*
Volume 5, Number 1-2, Pages 97–112

ROBINIA: SCALABLE FRAMEWORK FOR DATA-INTENSIVE SCIENTIFIC COMPUTING ON WIDE AREA NETWORK

YANG GU, GUOQING LI, QUAN ZOU, AND ZHENCHUN HUANG

Abstract. With the continuously growing data from scientific devices and models, data exploration becomes one of four kinds of scientific research paradigms. It leads to faster, larger-scale and more complex processing requirements, and parallelism is being more and more important for scientific data analyzing applications. But, because of troubles such as unstable wide-area network and heterogeneity among computing platforms, it is difficult to create scalable parallel scientific applications, especially wide-area parallel applications which have to process big data from geographically distributed research institutes to enable complex data analysis for "great challenge problems". In this paper, a data intensive computing framework named Robinia is proposed for exploiting parallelism among processing nodes over wide area network for data-intensive analysis on scientific big data. Robinia integrates distributed resources such as scientific data, processing algorithms, and storage services by a platform-independent framework; provides a unified execution environment for wide-area network based distributed spatial applications; and helps them exploit parallelism by a well-defined web-based programming interface. Experiments on prototype system and demo applications show that scientific analysis applications based on Robinia can achieve higher performance and better scalability by analyzing distributive stored big data over wide-area network such as Internet simultaneously.

Key words. parallel processing, wide area network, scientific computing, big data.

1. Introduction

In the last decades, more scientific devices are built and more scientific data are captured and stored. For example, devices such as the Square Kilometre Array of radio telescopes project, CERNs Large Hadron Collider, and astronomys Pan-STARRS array of celestial telescopes are capable of generating several petabytes (PB) of data per day, but present plans limit them to more manageable data collection rates. To enable the fourth paradigm for science based on data-intensive computing, tremendous capability and scalability are required for scientific data processing and visualization infrastructure to store, process, analyze and visualize the data so that information, knowledge and theories in the big data can be discovered and mined. In order to make the data processing faster, simpler and more scalable, parallelism is enabled for the scientific data analyzing applications, and one of the best ways to achieve scalable performance is exploiting parallelism of applications. It decomposes a data processing job on mass data into a lot of subtasks on distributed nodes, which will deal with a piece of data independently.

There are many ways for scientific applications to achieve parallelism. For example, High Performance Fortran (HPF) [1] and OpenMP [2] exploits parallelism by employing many processors or processor cores to process different parts of a single array, MPMD programming with MPI [3] extends the same idea to a distributed setting such as cluster, and a data-intensive distributed application may achieve it by running coarse-grain subtasks on geographically distributed computing nodes concurrently.

Received by the editors January 11, 2014 and, in revised form, March 21, 2014.
2000 *Mathematics Subject Classification.* 35R35, 49J40, 60G40.

Otherwise, scientific big data are often massive and geographically distributed among research institutes around the world. To analyze these data for some scientific results, applications must execute on heterogeneous computing nodes provided and managed by different agents. For example, a spatial application will query and analyze remote sensing data from several space agencies such as NASA, ESA and JAXA so that knowledge hidden in the big data set can be discovered. In this scenario, a coarse-grain parallel application can minimize the data transfer cost by scheduling subtasks to process data closer to where it is stored, and achieve better performance by allocating subtasks on more geographically distributed computing nodes. It is one of the best ways to achieve parallelism, especially on wide area networks (WAN) such as Internet.

But, it is challenging to create a scalable and efficient wide-area parallel application for scientific data processing, especially for beginners who have not much knowledge and experiment in parallelization for distributed context. Codes must be written carefully to decompose the processing job into subtasks, distribute data among nodes, transfer commands and messages through WAN, schedule subtasks for load balancing, monitor computing nodes, etc. So, programming models and frameworks which can help data-intensive application development and execution will be very valuable for scientific data processing.

Due to the lack of wide-area parallel processing framework for scientific data-intensive applications, we propose such a framework named Robinia, in order to enable parallel processing on wide area network for scientific data-intensive computing. First of all, distributed execution environment of Robinia exchanges commands and messages by standard protocols such as HTTP, so that firewalls can be passed through. Then, toolkits provided by Robinia such as dynamical node discovery, smart data distribution, description-based algorithm migration, and adaptive task scheduling, integrates increasing processing and storage resources all over the WAN together for distributed spatial applications. Furthermore, existing codes and algorithms for scientific data processing can be reused easily and deployed dynamically, and new codes and models can be developed simply by many programming languages and script languages. (e.g. Java, Beanshell, Groovy and Scala) As the result, much higher amount of scientific data can be processed by more computing nodes around the world without much performance loss and extra work when the problem size increases.

The paper is organized as follows. After the discussion about related work in section 2, section 3 proposes the architecture overview of Robinia, and designs components such as Executor, Engine, Node Discovery, Global Weather Focus and Web-based User Interaction for Robinia. Section 4 studies how to store mass data on a number of nodes by storage cluster, the atomic element for data storage in Robinia. Section 5 describes parallel processing implementation based on master-worker executors, and implements MapReduce model on Robinia as an example. Section 6 details the implementation such as event loop, distributed node discovery, and existing codes integration, and end-user interaction. Finally, section 7 discusses experiences about typical scientific data processing applications such as remote sensing parallel processing and distributed biological sequence comparing; and conclusions are summarized in section 8.

2. Related Work

There has already been a lot of work for exploiting data parallelism. Condor, which is renamed as HTCondor in [4], was one of the early examples of such distributed systems. When users submit their serial or parallel jobs to HTCondor, it places them into job queue, schedules them on computing nodes in a "Beowulf" cluster, monitors their progress and ultimately informs the user while completion.

Grid computing has been used to build operational infrastructures for data-parallel processing when it is proposed. Grid infrastructures are built to enable users the ability to access, modify and transfer extremely large amounts of geographically distributed data for research purposes by middleware and services, such as data transport, data access, data replication, task scheduling, and resource allocation. For example, projects such as G-POD from ESA [5] and GEO Grid based on Gfarm [6] are proposed for users to store and process earth observation data more on-demand and extensible based on grid infrastructures. In the recent years, more models and frameworks are proposed to exploit massive parallelism for scientific applications, such as MapReduce [7] and Dryad [8].

Googles MapReduce provides a good abstraction of group-by-aggregation operations over a cluster of machines. By a map function for grouping and a reduce function for aggregation provided by programmers, run-time system achieves parallelism by partitioning the data and processing different partitions on multiple machines concurrently. Apache Hadoop [9], an open-source Java implementation for MapReduce, is widely used for delivering highly-available services on top of a cluster of computers. However, this model has its own set of limitations. Users are forced to map their applications to the map-reduce model in order to achieve parallelism. This mapping is very unnatural for some scientific data processing applications. Furthermore, existing codes cant be deployed and re-used easily in MapReduce.

Microsofts Dryad is more general and flexible. It can execute arbitrary computation which is expressed as directed acyclic graph (DAG). A Dryad application combines simple computational vertices provided by users with communication channels to form a data flow graph, and runs by executing the vertices of this graph on a set of available computers, communicating through files, TCP pipes, and shared-memory FIFOs. But, Dryad is not appropriate for applications such as iterative jobs, nested parallelism, and irregular parallelism, which are frequently used in scientific computing. And, developing a new application by Dryad is still a hard job, especially for beginners.

Besides above productions, there are a lot of exploration projects on data-parallel processing models, services and infrastructures, such as Yahoo!s PigLatin [10], Microsofts SCOPE [11], Googles GFS [12], Big Table [13], Dremel [14], and Spanner [15]. Furthermore, much more achievements are proposed for data-parallel programming in some application domain, e.g. data parallel Haskell [16], Hyracks software platform [17], and GridBatch cloud computing system [18]. For example, to process remote sensing data parallel, the Matsu Project provide a cloud-based on-demand disaster assessment capability through satellite image comparisons [19], and Guan, et al. proposed a parallel framework for processing massive spatial data with a Split-and-Merge paradigm. [20] The Global Earth Observation System of Systems (GEOSS), which is managed by the Group on Earth Observations (GEO), an international collaboration of many organizations that produce and consume Earth observation data, deploys an international, federated infrastructure for sharing of Earth observation data products worldwide.

But, most of these productions, such as Hadoop, suppose that big data are stored and processed on local area network with high bandwidth and low latency. This questionable assumption makes data distribution and task scheduling much simpler. For example, transmission time for data distribution is predicable so that scheduler can optimize tasks better; processing elements can transfer commands and data by customized protocols for higher performance, etc. When these productions are migrated to WAN, the complex and unstable network environment invalids the assumption and brings much more troubles, especially to parallel processing across research organizations. For instance, transferring data from storage node to processing server may cost too long so that the benefit from parallel processing is spoilt; some services may be blocked by firewalls on the network border of institutes; processing nodes and storage nodes which are running a part of parallel job may crash or go off-line without any notification; and so on.

3. Architecture Overview

As a parallel processing framework for applications on the wide-area network, Robinia tries to exploit parallelism for processing scientific data distributed in institutes around the world with better performance, scalability and robustness. A center-less symmetric architecture is employed, in which nodes are independent and coequal for cooperation, no matter what they are desktop computers, high-performance clusters, or even super computers. On these nodes, a distributed platform-independent runtime environment middleware is installed so that nodes can discover each other for collaboration, and support the execution of wide-area data-intensive applications. The overview architecture is shown in figure 1.

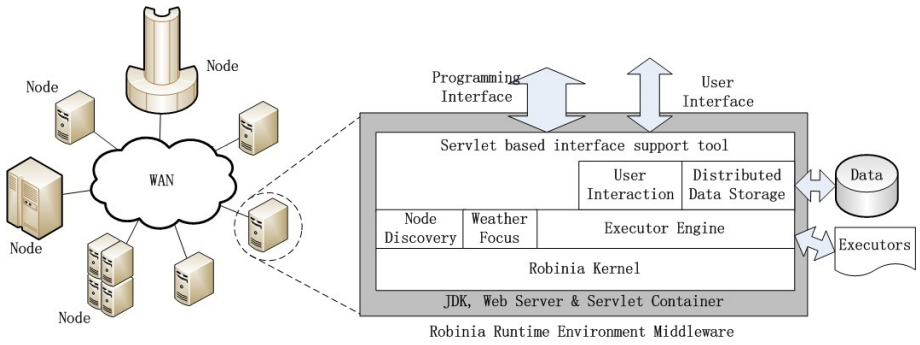


FIGURE 1. Architecture overview

Runtime environment employs an event-based asynchronous execution model for **Robinia Kernel**. Based on an event queue and a pack of event processor threads, Robinia kernel can provide functions such as event handling and timing for other modules in the runtime environment.

Since almost all firewalls on Internet allow clients inside to initiate HTTP or FTP connection to outside servers, it is logical to use these protocols to transfer commands, messages and data through the firewalls. A servlet based interface module receives and analyzes these encoded commands and messages, and dispatches them to the appropriate modules for further processing. Furthermore, a method named service reverse or active service is purchased to allow Robinia nodes inside to serve other Robinia nodes outside without any firewall reconfiguration.

In Robinia, all kinds of executable codes, including applications, are deployed and managed as **Executors** which described by XML-based configuration file shown in figure 2. Executors are hierarchical. An executor can invoke other executors locally or remotely. When an executor is usually invoked by user directly, it can be regarded as an application. **Robinia Engine** runs executors written in different languages, such as java, script language, native codes, workflow description languages, etc. Users can deploy their algorithms and methods simply by creating a configuration file. When required, executor engine will create a new instance by reference to the executor, start the instance, monitor its state transition, and collect its results for the invoker when instance accomplish. Furthermore, engine indexes deployed executors by their universal names and key-value properties for search.

```
<?xml version="1.0" encoding="UTF-8"?>
<executor>
  <name>echo</name>
  <type>java</type>
  <description>System executor for echo.</description>
  <feature name="class">System.System</feature>
  <parameter name="class">
    org.thgrid.robinia.system.SystemExecutors
  </parameter>
  <parameter name="method">echo</parameter>
  <auth name=".*">allow</auth>
</executor>
```

FIGURE 2. Configuration file for executors

To achieve parallel data processing on WAN, Robinia stores and manages the data to be processed by its **Distributed Data Storage**. By a well-defined programming interface, applications can store, search, and access distributed data with a hierarchic data model all over Robinia system. With different executor implementations, data can be stored temporarily in main memory, or permanently in SQL-based relational databases like mySQL [21] and no-SQL databases such as MongoDB [22].

Besides above, there are more modules in Robinia, such as Distributed Node Discovery, Global Weather Focus, and User Interaction Support, which provide more powerful services for applications based on Robinia. For example, **Node Discovery** tries to help center-less nodes discover each other via a set of boot-up servers; **Global Weather Focus** collects system information such as load of hosts and network weather regularly for better schedule, **User Interaction Support** provides a universal web-based user interaction framework for Robinia executors, and so on. Based on Java Development Kit and Servlet Container such as Apache Tomcat [23], Robinia runtime environment made up by these modules supplies capabilities to distribute, store, and process big data concurrently for high-performance and high-throughput data intensive applications.

4. Distributed Storage for Scientific Data

Robinia abstracts a data item as document with three possible parts: Meta data, values, and attachments. The meta data in a document is defined as the information about one or more aspects of the document, including global ID, usage, update

time, keywords about the data item, and so on. It can be regarded as the head of a data item. Values are located by storage reference in meta-data and stores the main part of the data item in nested key-value pairs as the body of a data item. (Shown in figure 3) It is the structured part of scientific data. Optional attachments referenced by attachment references in values are regarded as some attached data with huge amounts and usually stored as files and accessed as data streams without pre-defined structure. Applications can search data items by their meta-data, and access their values and attachment via its storage references in meta-data.

```
Record = {key : value, ...}
Key = string
Value = number|string|Boolean|list|record
List = {value, ...}
```

FIGURE 3. Schema of meta data and values

In Robinia, scientific data are stored in **Storage Clusters** which are the atomic elements for data storage. Storage cluster, shortly SC, is made up by two parts: one header node with fault-tolerant backups, and a pack of data nodes. It is shown in figure 4. Header node is the main entrance of a storage cluster; it stores and indexes meta data for all data items in the storage cluster. While an application requires scientific data, it should search and get meta data from header node first, and then accesses the values and attachments on data nodes under the guidance of storage reference in meta data. A data item may has more than one copy of values and attachments referenced by storage reference in meta data, so that the storage cluster can access it free when some nodes are down or off-line.

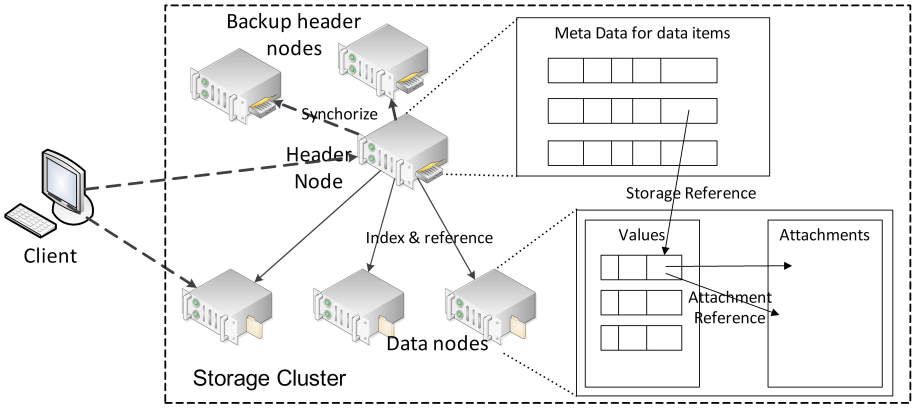


FIGURE 4. Storage Cluster

Based on mongoDB, a no-SQL database, two executors named SCHeader and SCData are designed and implemented for header nodes and data nodes in storage cluster. They receive XML based encoded requests for data query or data access, and send XML based responses while the requests are processed. Supported by a client library, applications can invoke these executors to search meta data, access data values, and stream attachments locally or remotely.

5. Implementation of Parallel Processing

To process scientific data parallel, distributed application should deploy a master executor and a series of worker executors in Robinia. In these executors, master plays the role of controller and scheduler, which schedules workers with same or different algorithms and implementations on different platforms to process the input and immediate data set.

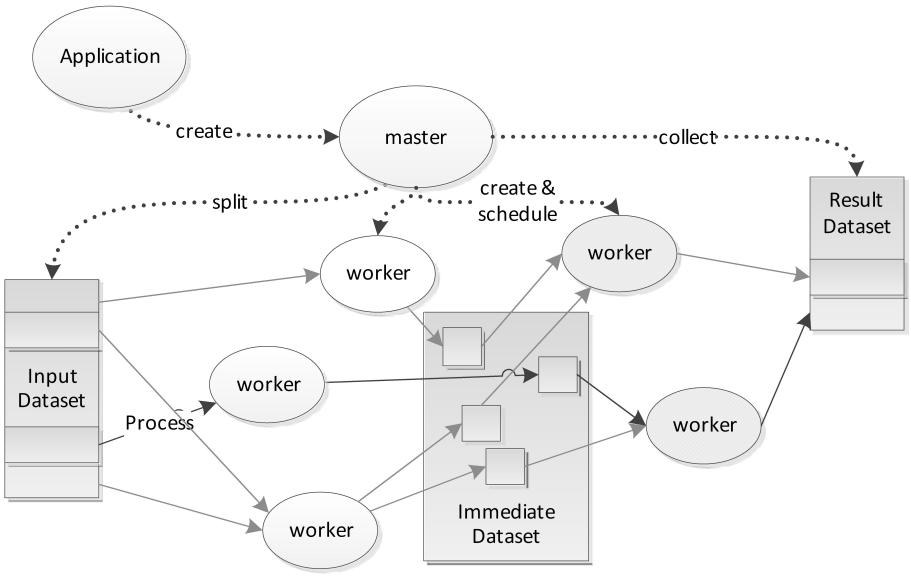


FIGURE 5. Execution of distributed applications

As shown in figure 5, data-intensive distributed applications process scientific data by following sequence of actions.

(1) Master splits the processing task into pieces by partitioning input data set first. When the application initializes, input data set may be already distributed in Robinia SC, or still stored out of Robinia. Master just splits input data set into pieces logically without any data transfer.
(2) Master finds available computation nodes for workers, and schedules workers to process partitioned data sets on them. Worker executors are created as close as possible to where data stored, so that it will cost less time for workers to transfer input or immediate data.
(3) Workers are monitored so that master can collect their result data set when process complete, if it is necessary. Furthermore, when some workers are off-line because of accidents such as software fault or network failure, master will schedule other workers to take their place for continuous execution of distributed application.

Based on well-defined master and worker executors, MapReduce can be implemented, too. In Robinia, a MapReduce application will be launched by starting a MapReduceMaster executor which commands three different kinds of workers: mapper, shuffler, and reducer. It is shown in figure 6.

When MapReduceMaster is started, it splits input dataset into pieces and finds available processing nodes to schedule mappers provided by application developer for map function on these pieces of data. When all mappers are accomplished, a

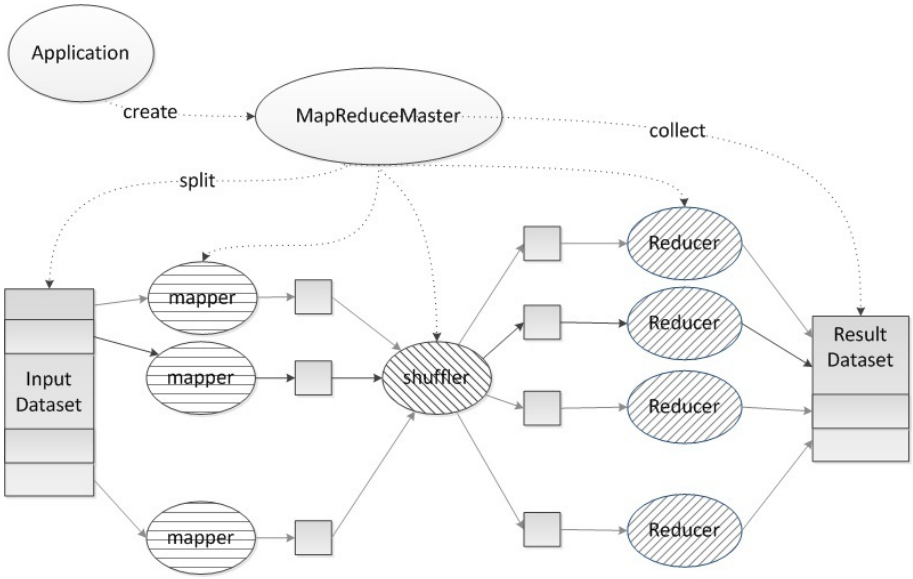


FIGURE 6. MapReduce based on Robinia

shuffler executor will be started to shuffle results of mappers for reducers. Then, reducers run reduce functions for pieces of result dataset which will be collected by MapReduceMaster for the final result. Different from mapper and reducer which should be provided by applications developer, shuffler can be chosen from those provided by Robinia, as well as implemented for customized data shuffling or better performance.

6. Implementation Details

For its performance and platform-independence, Robinia adopts Java language to implement its runtime environment. And the run time environment is released as a WAR package which can easily deployed in Java Servlet container such as Apache Tomcat 6.0.x or higher.

In its kernel, Robinia runtime environment employs an event-based asynchronous execution model. Requests such as create an instance by given executor or run a benchmark are posted into event queue as an event for an event loop thread to peek them. When a new event is found in queue, event loop thread selects an event processor thread from thread pool, and dispatch the event to it for handling the event. Event is also used for timing by timing event which can be posted on a given time.

As a distributed system, it is very important for Robinia to support its nodes discover each other without single point of failure. In Robinia, each node must send a set of registry servers heart-beat data to keep alive and refresh the current available node list regularly. If heat-beat is not heard from some nodes for a period of time (time-out), registry server will remove these nodes from its node list. To avoid single point failure, nodes in Robinia vote multiple registry servers automatically, so that distributed node discovery can keep work when some of the registry server nodes crash. Robinia node can make node list as complete as possible by merge lists got from different registry servers.

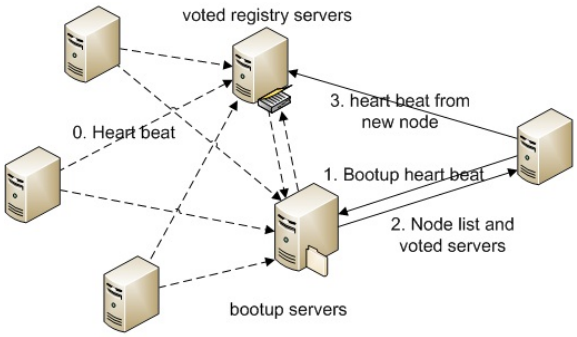


FIGURE 7. distributed node discovery

As shown in figure 7, a Robinia node must register itself by sending a boot-up heart beat to some manually configured boot-up servers first when it initializes. While node list and voted server list responded by boot-up servers is received, the new node will start to send heart-beat to voted registry servers regularly to keep it from being removed. In most scenes, boot-up servers are usually voted as registry servers for better performance.

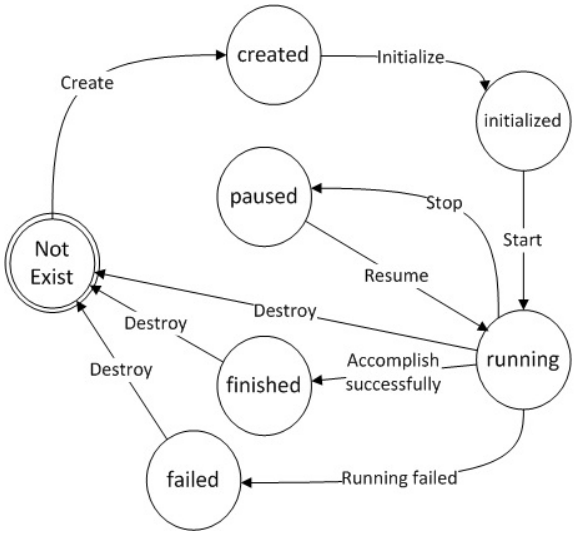


FIGURE 8. states of an instance in a life-cycle

In Robinia, executors can be programmed by java class, beanshell script, groovy script, scala script and native codes on processing nodes, and they are described by a XML based description file for their interfaces and implementations. Benefit from the plug-in based executor running environment, more programming languages and script languages will be supported to develop executor in Robinia soon. At the same time, existing codes and algorithms can also be deployed as executors by writing a description file easily, so that accumulated codes and algorithms will go on serving with less modification. All running executors are called instances which can be created when needed. If it is an instance of on-demand executor which algorithm should migrate from other nodes, Robinia must initialize the instance by migrating

proven algorithm from its provider first. Then, the instance can be started to process or query data. When it is running, Robinia will keep monitoring its states, until it is accomplished and destroyed. Figure 8 shows the states of instances in a life cycle.

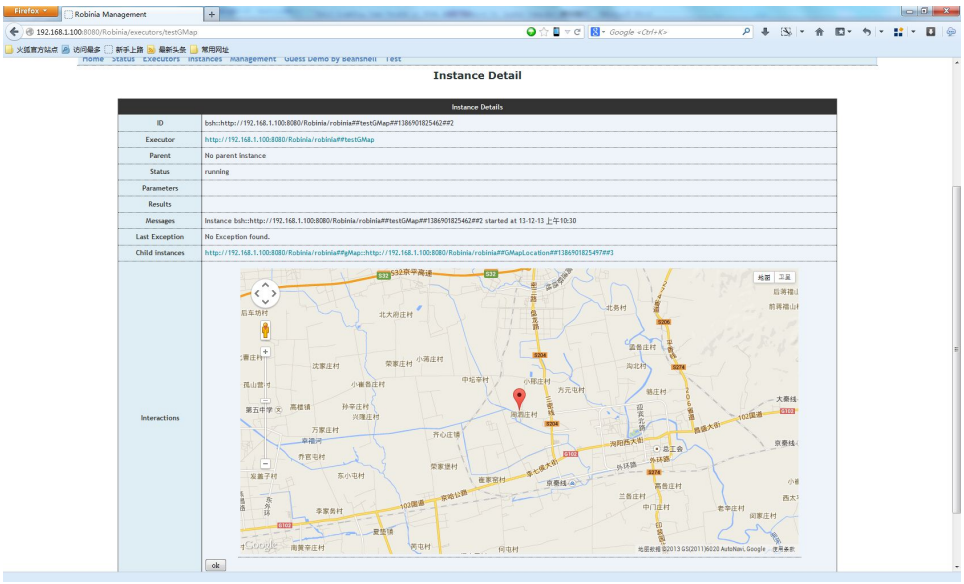


FIGURE 9. Example of web based user interface

Applications often need interact with users when running, so that users can control the execution of applications, such as view the immediate or final result, input parameters, select target data set, etc. Some pre-defined executors with feature *class=WebUI.** can be invoked for applications to interact with users via web pages. For example, an executor with name WebInput can be invoked to show some messages to user and get a response string from user by an embedded JSP page. Application developers can implement new interaction executors by embedding some standardized JSP page fragments. Figure 9 is an example of web based user interaction page with google map embedded.

7. Experiences and Discussion

Based on Robinia, we build a distributed data-intensive scientific processing prototype, and deploy sample applications for remote sensing data analysis and biological sequence comparing. The applications are benchmarked for their performance and scalability on the prototype assembled by four PC nodes with Intel Core i3 @ 2.93GHz CPU, 4GB memory, 1 TB 7200rpm hard disk and windows 7 64-bit system connected with Giga-bits Ethernet.

7.1. Experiences about Remote Sensing Data Analysis. To make the application development easier, frequently-used tools such as the MODIS Reprojection Tool [24] are adapted as executors in the prototype for some basic functions such as resampling and image format conversion. As an example, we develop an application for global drought detection by Normal Differential Water Index (NDWI) brought up by Gao in 1996 [25]. It can be calculated by:

$$NDWI = (\rho_2 - \rho_6)/(\rho_2 + \rho_6)$$
$$avgNDWI = \sum_{i=1}^{n} NDWI_i/n$$
$$AWI_i = NDWI_i - avgNDWI$$

Here, NDWI is the difference between two bands in MODIS data, green(ρ_2, 0.86-μm) and nearinfrared (NIR, ρ_5, 1.24μm). avgNDWI is the average value of NDWI in given time scope. AWI is short for Anomaly Water Index, which points out how drought the vegetation canopies are in a given time.

We implement the application based on the Java HDF Interface (JHI) [26] from the HDF Group. To benchmark performance and scalability of the prototype, we run the application on 4 test data sets with different sizes for evaluation. They are shown in table 1.

TABLE 1. Data sets for global drought detection tests

Test No.	Data size	Descriptions
1	23GB	A small MODIS data set for test
2	180GB	MODIS data on the same day in eleven years
3	361GB	MODIS data on the same two days in spring and autumn of eleven years
4	722GB	MODIS data on the same four days in all seasons of eleven years

In the global drought detection application, all NDWI indexes for the same area will be averaged for AWI indexes calculation. If image data set for the same area is distributed on multiple storage nodes, additional time cost is necessary for collecting NDWI index data items for AWI index calculation. To reduce the time cost of data transfer, MODIS image data sets which cover the same area are stored in the same data storage node in advance. It is an example for Smart data partition. Figure 10 shows the result of AWI indexes on March 5th, 2002 as an example. It is a part of test 4. All processing results from parallel application are the same as them from the original serial application.

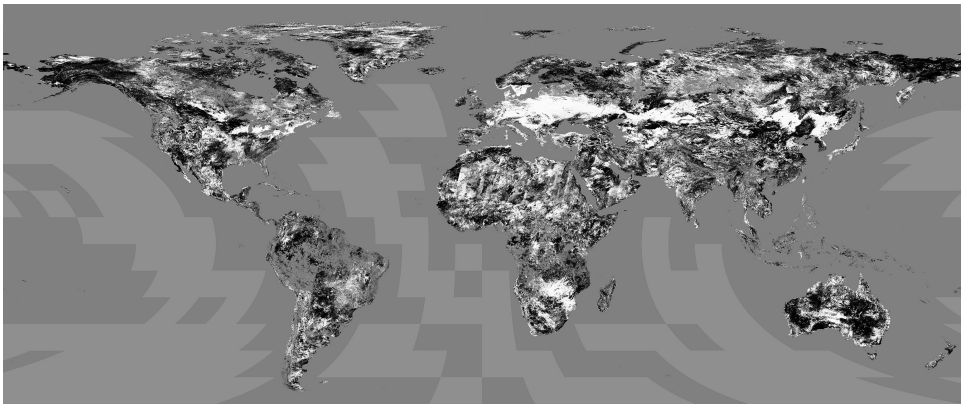
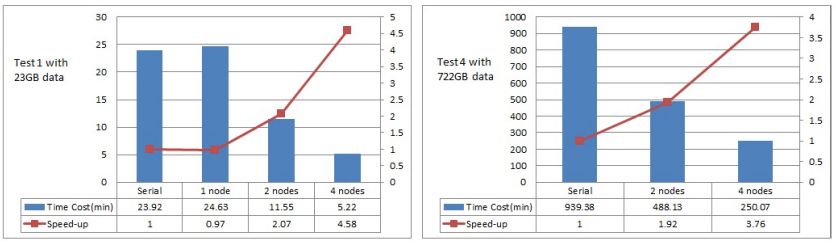


FIGURE 10. AWI indexes on March 5th, 2002 as result of the test application

To study the performance and scalability of Robinia, we benchmark the parallel application and the serial application on test cases shown in table 1. First of all, we run the application on 1, 2, and 4 computation nodes to process test data set 4 which is already partitioned and stored on the processing nodes. At the same time, we run the serial application on a PC with the same configuration for reference. The time cost and speed-up are shown in figure 11.



(a) Result for test 1 (b) Result for test 4

FIGURE 11. Time costs and speed-ups of Remote Sensing Data Analysis Benchmark

Figure 11 shows that the speed-up factors on 4 nodes are 4.58 (for test 1) or 3.76 (for test 4), and their efficiencies are 4.58/4=115% or 3.76/4=94%. The quasi-linear to super-linear speed-up results show that Robinia is high performance, and scalable on node size. Furthermore, we run benchmarks on all the test data sets for the data size scalability of Robinia, and the results are shown in figure 12. In the result, average time costs for processing 1 GB input data are 5.22/23=0.227minutes, 54.69/180.5=0.303minutes, 132.3/361=0.366minutes, and 250.07/722=0.346minutes. It suggests that the time cost is almost linear with the size of data set, except test data set No. 1, which is so small that memory cache for disk I/O plays a very important role and accelerates the processing much more.

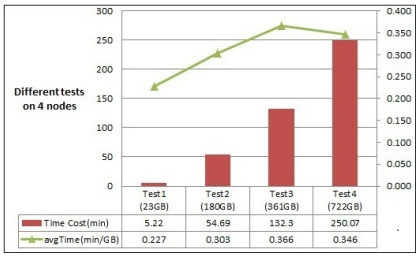


FIGURE 12. time costs on different data sets

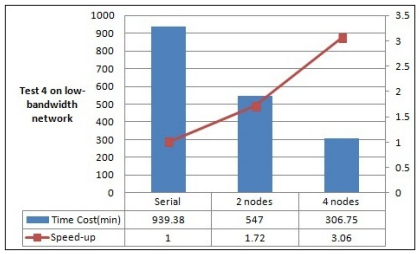


FIGURE 13. time costs and speed-ups on low-bandwidth network

In order to evaluate the performance of the prototype on WAN, we migrate the prototype to a low-speed network on which all computation nodes share 10Mbps bandwidth. Benchmark result on test case 4 is shown in figure 13. Unsurprisingly, the parallel processing efficiency is reduced for about 18% because it costs more time for transferring commands and messages on WAN. But, speed-up over 3 on

four nodes prototype indicates that Robinia is still scalable and extensible on low-bandwidth network such as WAN.

7.2. Experiences about Biological Sequence Comparing. When a DNA or protein sequence is found, comparing it with a database of known sequences to find similarities is one of the most important ways to study functions and features of this new sequence. BLAST (Basic Local Alignment Search Tool) [27], provided by NCBI for aligning query sequences against those present in a selected target database, is one of the most important tools in biological research. Based on Robinia, we deploy a parallel BLAST on wide-area network and schedule multiply computing nodes to compare sequences simultaneously for better performance.

There are several approaches for the parallelization of BLAST, such as partition the database and split the query. In this paper, we partition the database by command line tool makeblastdb to parallel BALST, and run distributed applications which adapt blastx and blastp command line tools on our 4-nodes prototype. The nr database which is the most popular protein database in NCBI was chosen as the testing database, and two difference sequence sets are used as query sets for test. (table 2)

TABLE 2. Test case for Biological Sequence Comparing

Database	nr (32052081 sequences and almost 11.12GB until 2013.08.26
Query set 1	One DNA sequence of 14000 byte length from before research
Query set 2	20 protein queries selected from Arabidopsis thaliana, with size from 129bp to 2214bp

The parallel BLAST application returns the sample result as the serial one, and runs much faster than the serial one. On our 4-nodes prototype, the speed-ups of parallel BLAST for query sets above are 3.52 (for query set 1) and 4.18 (for query set 2), and the efficiencies are 3.52/4=88% and 104.5% (Figure 14). They are also quasi-linear or super-linear, and show that Robinia is also high performance, scalable and extensible for Biological Sequence Comparing applications.

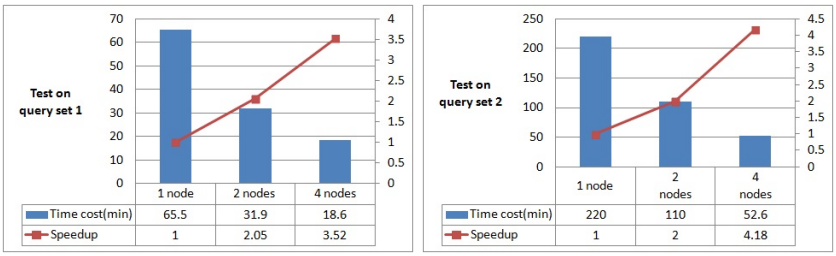


FIGURE 14. time costs and speed-ups of Biological Sequence Comparing benchmark

From the experience above, we find that Robinia can exploit data parallelism for scientific data processing on wide area network such as Internet, and endows data parallel applications scalable performance. It is very valuable for those scenes that

scientists need applications to process big data stored in different research institutes. With more tool executors, data prefetching, processing algorithm migration, and adaptive scheduling methods, applications based on Robinia can achieve higher performance, better scalability and fault-tolerance for more application domains.

8. Conclusion

In this paper, we try to exploit parallelism for scientific data processing on wide area network such as Internet based on Robinia, a light-weight, platform-independent and high performance data-intensive scientific computing framework. First, standard protocols such as HTTP are adopted to transfer commands and messages through firewalls all over the Internet, and service reverse enables nodes inside to serve clients outside without re-configuration of firewalls. Based on them, nodes with Robinia runtime environment can discover each other for collaboration simply and dynamically.

Second, storage cluster which is the basic element for scientific data storage is proposed. It stores data in three parts: meta data, values, and attachments. Users can query meta data stored on the header node by pre-defined properties such as sensor name, acquirement time, area covered, and band type for remote sensing data as a sample. Referenced by the meta data, values and attachments stored on data nodes can be accessed via their storage references and attachment references.

Furthermore, parallel processing on remote sensing data is supported based on Robinia executors which play roles of master and worker. Under the control of master, workers complete the decomposed process jobs on partitioned data simultaneously for better performance and scalability. Based on executors named MapReduceMaster and shuffler provided by Robinia, developers can implement and deploy MapReduce applications on Robinia easily by develop a map function and a reduce function. At the same time, existing codes and algorithms can also be deployed as executors by writing a description file easily, so that accumulated codes and algorithms will go on serving with less modification.

Finally, a distributed data-intensive processing prototype for scientific data is implemented and applications for global drought detection by NDWI and Biological Sequence Comparing by BLAST are developed and deployed. Tests with different input data sets and network conditions are launched for performance evaluation. The results with quasi-linear to super-linear speed-up factors shows that distributed applications for scientific data processing supported by Robinia can achieve high performance simply, elastically, fault-tolerant and platform-independent.

Acknowledgments

The author thanks the anonymous authors whose work largely constitutes this sample file. The research is supported in part by National High-Tech Research and Development Program of China (863 Program) under Grant No. 2011AA120306, and by National Natural Science Foundation of China (NSFC) under Grant No. 60703054.

References

[1] Loveman D B.: High performance fortran. Parallel & Distributed Technology: Systems & Applications, IEEE, 1993, 1(1): 25-42.

[2] Dagum L, Menon R.: OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE, 1998, 5(1): 46-55.

[3] Message Passing Interface, http://www.mcs.anl.gov/research/projects/mpi/

[4] Douglas Thain, Todd Tannenbaum, and Miron Livny.: Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience, 17(2-4):323C356, 2005
[5] Cossu, Roberto, et al. ESA Grid Processing on Demand for fast access to Earth Observation data and rapid mapping of flood events. European Geosciences Union General Assembly (2008).
[6] Sekiguchi S, Tanaka Y, Kojima I, et al. Design principles and IT overviews of the GEO Grid. Systems Journal, IEEE, 2008, 2(3): 374-389.
[7] Jeff Dean and Sanjay Ghemawat.: MapReduce: Simplified data processing on large clusters. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI), pages 137C150, December 2004
[8] Isard M, Budiu M, Yu Y, et al.: Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Operating Systems Review, 2007, 41(3): 59-72.
[9] Welcome to Apache Hadoop!, http://hadoop.apache.org/
[10] Olston C, Reed B, Srivastava U, et al.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008: 1099-1110.
[11] Chaiken R, Jenkins B, Larson P Å, et al.: SCOPE: easy and efficient parallel processing of massive data sets. Proceedings of the VLDB Endowment, 2008, 1(2): 1265-1276.
[12] Ghemawat S, Gobioff H, Leung S T.: The Google file system. In: ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.
[13] Chang, F., Dean, J., Ghemawat, S., et al:, Bigtable: a distributed storage system for structured data, OSDI 2006, 205-218.
[14] Melnik S, Gubarev A, Long J J, et al.: Dremel: interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment, 2010, 3(1-2): 330-339.
[15] Corbett, J. C., Dean, J., Epstein, M., et al.:, Spanner: Googleś Globally-Distributed Database. In: Proceedings of the 10th USENIX Symposium on Operating System Design and Implementation (OSDI Í2), 2012: 251-264.
[16] P.W. Trinder, H-W. Loidl, and R.F. Pointon: Parallel and distributed Haskells. Journal of Functional Programming, 12(4&5):469C510, 2002
[17] Borkar, V., Carey, M., Grover, R., Onose, N., Vernica, R.: Hyracks: A flexible and extensible foundation for data-intensive computing. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pp. 1151-1162
[18] Liu, H., Orban, D.: Gridbatch: Cloud computing for large-scale data-intensive batch applications. In: Cluster Computing and the Grid, 2008. CCGRIDÓ8. 8th IEEE International Symposium on, pp. 295-305
[19] D. Mandl, Matsu: An elastic cloud connected to a sensor web for disaster response, in Ground System Architectures Workshop (GSAW), Workshop on Cloud Computing for Spacecraft Operations, Mar. 2, 2011
[20] Guan X, Wu H, Li L. A Parallel Framework for Processing Massive Spatial Data with a SplitCandCMerge Paradigm. Transactions in GIS, 2012, 16(6): 829-843.
[21] MySQL :: The world's most popular open source database, http://www.mysql.com/
[22] MongoDB, http://www.mongodb.org/
[23] Apache Tomcat, http://tomcat.apache.org/
[24] MODIS Reprojection Tool LP DAAC — LP DAAC :: ASTER and MODIS Land Data Products and Services, https://lpdaac.usgs.gov/tools/modis_reprojection_tool
[25] Gao B C.: NDWIa normalized difference water index for remote sensing of vegetation liquid water from space. Remote sensing of environment, 1996, 58(3): 257-266
[26] McGrath, Robert E., Xinjian Lu, and Michael Folk. Java (TM) applications using NCSA HDF files. Concurrency Practice and Experience 9.11 (1997): 1113-1125
[27] Altschul S F, Gish W, Miller W, et al. Basic local alignment search tool. Journal of molecular biology, 1990, 215(3): 403-410

Department of Computer Science and Technology, Tsinghua University, Beijing, China
E-mail: guyang3532@qq.com

Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing, China
E-mail: gqli@radi.ac.cn

Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing, China
E-mail: qzou@radi.ac.cn

Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
E-mail: huangzc@tsinghua.edu.cn