Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 9 (3): 686-701, 2015
ISSN 1819-4311 / DOI: 10.3923/jse.2015.686.701
© 2015 Academic Journals Inc.

Research on the Architecture of Data-Intensive Computing Platform

[1,2]Ke Hou, [1]Jing Zhang and [2]Xing Fang
[1]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, 710048, China
[2]School of Economic and Management, Xi'an Shiyou University, Xi'an, 710065, China

Corresponding Author: Ke Hou, School of Economic and Management, Xi'an Shiyou University, No. 18 East 2th Dianzi Road, Xi'an, Shaanxi, 710065, China Tel: +86 29 8838 2653

ABSTRACT
Data-Intensive Computing (DIC) is a kind of parallel computing which is specific to massive, distributed, heterogeneous and changing dataset processing. The architecture of DIC platform is a set of multiple abstract models, these models describes the function compositions, characteristics, coupling relationships, interaction ways and application scope of each layer in DIC platform. This article studies the architecture of DIC platform. Firstly, the architecture of related parallel computing platform is reviewed. Secondly, the design requirements of DIC platform are analyzed, the integrated research method of DIC is discussed and then an architecture of DIC platform with seven layers is provided. Finally, in order to verify its feasibility and effectiveness, a simple prototype system is implemented to support mass image data parallel processing. Compared with serial processing mode, the prototype system can obtain higher speed-up.

Key words: Data-intensive computing, architecture, integrated research method, mathematical morphology

INTRODUCTION
Computer technology is a modern and intelligent way of information collecting and data processing, it has been used in many fields. However, with data volume increasing rapidly (Winter, 2008), the traditional data analysis and processing system cannot adapt to this trend of evolution. On the one hand, a lot of storage and computing resources are occupied and the amount of concurrent tasks and users is very large. On the other hand, it took too much time that required data are accessed for one computing task, while for many computing tasks, requirement of the timeliness aggravated this problem. The "big data" problem results in Data-Intensive Computing (DIC) be proposed and become the focus of industry, scientific community and computer academia (Howe *et al*., 2008).

Data-intensive computing is not a brand new technology, it is closely related with parallel computing, distributed computing, High Performance Computing (HPC) and cloud computing (Hayes, 2008). Fundamentally, DIC belongs to parallel computing and its underlying thought is to decompose computing problem into many tasks which will be delivered to multiple computing resources and executed in the same time. This happens to coincide with the current requirement of massive data processing.

Data-intensive computing is a kind of parallel computing which is specific to massive, distributed, heterogeneous and changing dataset processing. Except analyzing and understanding large-scale data, DIC should also be responsible for accessing and maintaining these dataset. While thinking about this definition, it firstly means that the DIC task are all developed around the data which is not only large-scale but also distributed, heterogeneous and changing. Secondly, it means the DIC task refers to the whole processes which range from data access to data storage, analysis, processing and understanding. The third meaning is the new characteristics of DIC make the traditional data management technologies be no longer fit to be used.

Different with the traditional HPC, DIC focuses on large-scale data which has massive volume and the data itself is distributed, heterogeneous, unspecific and inconsistent. These make DIC similar to cloud computing. On the one side, the cloud services which appear later are developed to satisfy the new requirement of DIC initially and the storage and calculation are common things between cloud computing and DIC. On the other side, cloud computing has clear apply background and significant business value and important characteristics (e.g., resource pool, resource as a service, payment on demand). The cloud can provide elastic, pay-on-demand information infrastructures for DIC systems including storage, calculation and network resources.

For the user, the target to upload and execute DIC tasks is to process, analysis and understand large-scale, distributed and changing dataset. The DIC platform is a distributed parallel computing system aiming at large-scale data storage and processing. Hence, designing DIC platform aims at integrating soft and hard resources of data center, sharing and configuring these resources, thereby providing the important guarantee to solve large-scale data processing problem in network information service, science and engineering calculation and electronic commerce fields and improving significantly the processing efficiency and reducing the computing cost.

The architecture of DIC platform is a set of multiple abstract models which describe the function compositions, characteristics, coupling relationships, interaction ways and application scope of each layer in DIC platform. Analyzing and designing the architecture of DIC platform has higher theoretical significance and strong practical guiding value.

ARCHITECTURE OF RELATED PARALLEL COMPUTING PLATFORM

Cloud computing platform: Cloud computing is a kind of distributed computing model, it make calculation tasks be distributed in a resource pool which consists of a large number of computers and user can obtain computation, storage and information services on demand. Usually, cloud computing can be divided according to service type and service object. Due to different service types, the Infrastructure as a Service (IaaS), the Platform as a Service (PaaS) and the Software as a Service (SaaS) are three kinds of cloud computing service (Luo *et al*., 2011). The IaaS layer encapsulates infrastructure resources (e.g., hardware equipments) as service, such as Amazon EC2, Amazon S3, Eucalyptus. The PaaS layer takes further abstract of resources to provide development and execution environment for user application, for example, Google App Engine, Hadoop, Microsoft Azure. The SaaS layer encapsulates specific software function as a service (Salesforce CRM, Google Apps, etc.). According to different service objects, cloud computing can be divided into public cloud, private cloud, hybrid cloud and cloud community (Marinos and Briscoe, 2009). Different with other views, the researchers of UC Berkeley thought that cloud computing is general term of SaaS and utility computing, excluding private cloud (Armbrust *et al*., 2009). After two main cloud computing frameworks are analyzed and other domestic and overseas solutions are summarized, a cloud computing architecture is proposed, as shown in Fig. 1.

HPC platform based on grid: The HPC platform based on grid is the supporting environment of integrating HPC resources which includes system hardware, system software, software tools and management systems and so on. The grid computing went through a development process from computing oriented to integration of computing and service. One of the main characteristic of grid

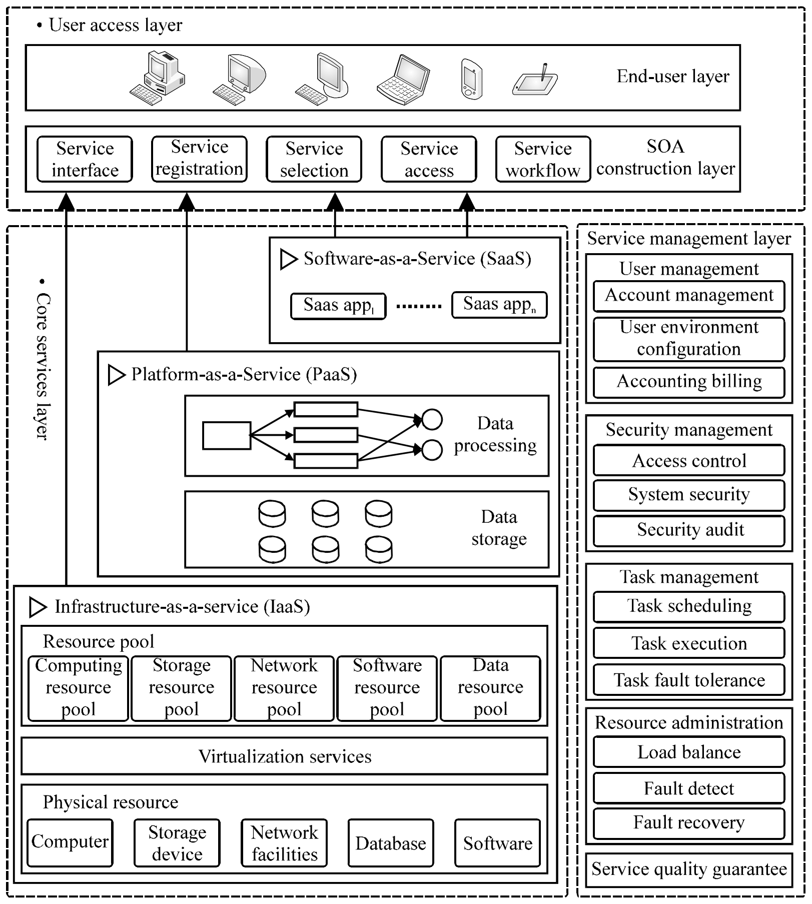

Fig. 1: Cloud computing architecture

computing is to solve the internet-oriented resource sharing and collaboration problem and the developing trend of its architecture is a open architecture based on service. This HPC-oriented grid architecture should not only satisfy the requirement of HPC but also be easy to manage, configurate and maintain. Finally, it should reflect the advantages of HPC and meet application, function and performance requirement to application service of the platform.

To achieve the virtual organization management and resource sharing, the academia had gotten many achievements in grid architecture. Among of those achievements, two grid architectures are representative: One is the five-level sandglass grid architecture represented by Foster *et al.* (2001), as the Fig. 2 shows. The other is proposed by IBM and Ian Foster together, called the Open Grid Services Architecture (OGSA) (Foster *et al.*, 2003).

Existing DIC architectures: Dig data is the current issue of DIC, in order to implement complex data-intensive tasks in real-time, on the base of the theory of Marz and Warren (2013), Garber (2012) and Chen and Zhang (2014) proposed the seven necessary principles to guide the big data analytics systems designing, the first of which is that good architectures or frameworks are crucial.

To solve different DIC problems, several architectures were proposed. Hyracks is a new partitioned-parallel software platform for DIC, it can run on the large cluster which is not shared

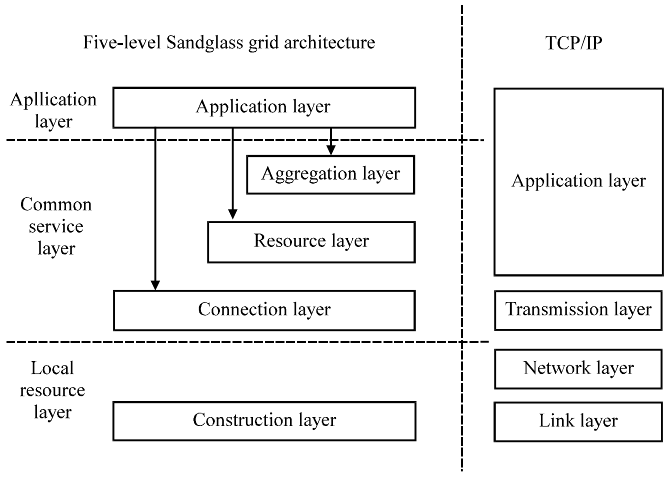

Fig. 2: Five-level sandglass grid architecture and TCP/IP

(Borkar *et al.*, 2011). Hyracks allows users to express a computation as a DAG for data operation and connection. To deal with the DIC problems of scientific workflows, some reformative scientific workflow management system was proposed, for example, Kepler+Hadoop in which researchers can utilize easily MapReduce in their domain-specific problems and connect them with other tasks in a workflow through the Kepler graphical user interface (Wang *et al.*, 2012a). The Lambda architecture (Marz and Warren, 2013) aims at solving the general data system problem on which arbitrary functions can be implemented on arbitrary data. Lambda decomposes the robust and scalable data system into three layers: The batch layer, the service layer and the acceleration layer. To deal with semantics data, Kourtesis *et al.* (2014) proposed a semantics-oriented quality of service management architecture based on Lambda architecture which utilized advantages of the semantic technology and the distributed data stream processing.

METHODOLOGY

Research method and system requirement

Integrated research method of DIC platform: Looking back upon the development of parallel computing, the problems and research achievements show that establishing a complete scientific research system is crucial. Chen *et al.* (2008) proposed the theory of parallel algorithm research system which means "theory-design-implementation-application". In 2009, he put forward the integrated research method of parallel computing named "structure-algorithm-programming-application" (Chen *et al.*, 2009). Although, the simplified data management steps of the widely-used distributed computing architectures (e.g., MapReduce and Dryad) have shown strong data management ability. Just as Hayes (2008) pointed out, these simplified distributed computing architecture is not the best choice. After introducing some new features (e.g., appropriate indexes), system performance may be increased significantly. Guo and Xu (2011) thought that the GPU should be introduced to the framework, in order to meet the mixing needs which are combination of computational-intensive and data-intensive needs.

Fundamentally, DIC is a kind of parallel computing. In this study, the integrated research method of DIC platform which is called "system structure-data acquisition-data storage-data processing-upper application" comes from the new characteristics and problems of DIC and it developed previous research achievements as shown in Fig. 3.

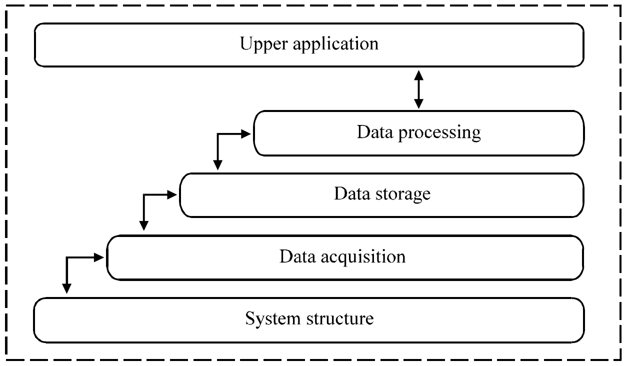

Fig. 3: Integrated research method of DIC platform

Requirement of DIC platform

Requirements on scalability: Facing massive data, the DIC platform should ensure the high scalability of data management and processing and allocate and recycle resources in accordance with workload. The DIC application should also adapt to changing resource and response to specific situation. The DIC platform should use the judgment and prediction tool for resource dynamic scaling, so the resources scale can be adjusted automatically and rapidly according to workload of the upper application, so as to satisfy the scalability requirement of DIC application with increasing data volume in many respects (response time, throughput and running cost, etc).

Requirements on managing heterogeneous data: Usually, DIC application would get data from different data sources, so the data is likely to be heterogeneous, such as structured, semi-structured, unstructured. To deal with heterogeneous data, DIC platform should have comprehensive abilities to retrieval, query, store, extract, analyze, process and mine heterogeneous data and different DIC applications may require different programming models to support the same data processing algorithm which is important challenge faced by DIC platform.

Requirements on calculation tasks diversification: Types of DIC tasks are diversification, such as embarrassingly parallel calculation, iterative calculation, multi-dataset processing and complicated calculation with dependency. There are large differences among these calculation tasks, so the processing logic of parallel algorithms has very strong pertinence. Because implementing parallel algorithms would adopt different parallel programming methods, different parallel programming models are involved. This is a challenge to DIC platform while it integrates multiple programming models and computing framework and ensures high usability.

Requirements on system manageability: In the future, DIC platform need to provide unified operation and supporting environment for a series of same or different types of application. Some applications are developing gradually to multi-tenant application mode. The management model which includes multiple elements (e.g., application, user, session, resources and job) will get more and more complicated. The demands for multi-application management will also show great differences which involve system deployment, system configuration, load balancing, fault tolerance and performance tuning and so on. DIC platform has urgent need to optimize and improve its manageability, even redesign and restructure itself.

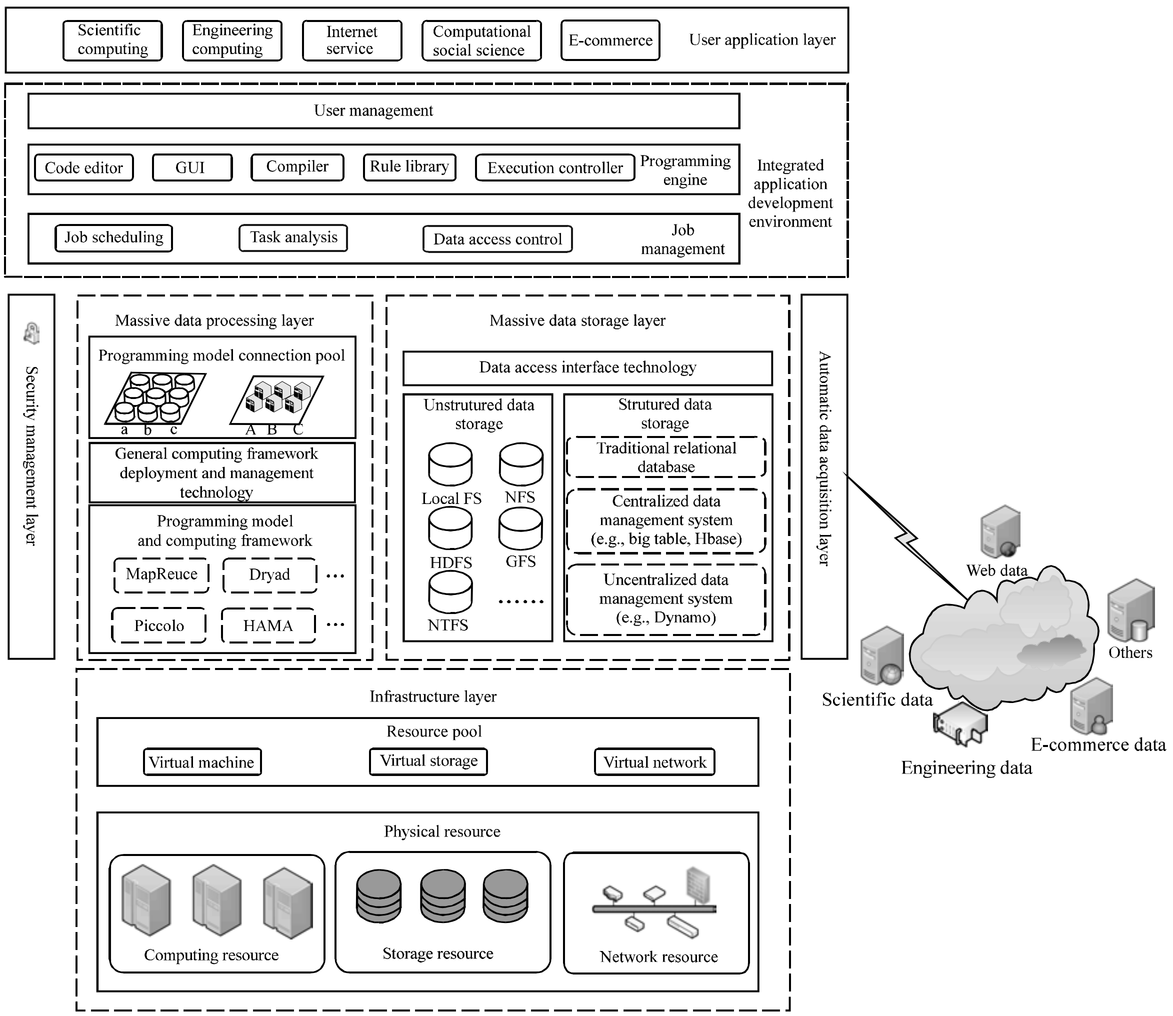

Fig. 4: Architecture of DIC platform

Requirements on fault tolerance: Usually, DIC platform is deployed on the large-scale, distributed cluster. In order to decrease cost as much as possible, the cluster is composed of cheap business machines. Therefore, the node failure and data failure of cluster are regarded as common phenomenon. While designing DIC platform, in order to reduce the overhead of redoing some subtasks and even the whole operation a good fault tolerance mechanism for application should be provided, such as copy management, checkpoint strategy. To ensure the availability of DIC platform, data consistency can be even sacrificed to a certain extent.

DESIGNING ARCHITECTURE OF DIC PLATFORM

According to target and requirement of designing DIC platform, this study applies the integrated research method oriented to DIC and puts forward the architecture of DIC platform with seven layers which includes user application layer, integrated application development environment, security management layer, massive data processing layer, massive data storage layer, automatic data acquisition layer and infrastructure layer, as shown in Fig. 4.

User application layer: This layer includes all kinds of data-intensive applications which are developed and deployed by related companies and institutions. These companies and institutions can be seen as data-intensive application providers, the service objects (i.e., end users) could be external users (e.g., internet service users), also could be internal users (e.g., users of scientific and engineering computing application). The DIC platform is transparent for end users, who acquire and utilize all kinds of data-intensive applications and services but they do not need to concern how to deploy and manage application, or to implement the underlying software.

Integrated application development environment: The integrated application development environment provides a unified interface for data-intensive application provider (i.e., DIC user) and it is safe, efficient and elastic development environment and assistant tools. The integrated application development environment includes three sub-layers. The first sub-layer is user management, it takes charge of creating, updating and deleting users and assigning and controlling permissions. The user includes developer (i.e., data-intensive application provider) and manager (i.e., the DIC system administration). The second sub-layer is programming engine, it is the main controller and user interface of the entire data processing layer and it is responsible for uploading, editing, transforming, starting and stopping source codes. This sub-layer also takes charge of initialization, configuration, maintaining and management of programming models in the rule library and it controls the whole system. The third sub-layer named job management is in charge of transforming an application into a job and then decomposing it into some sub-jobs. This sub-layer is able to append and update the related information in the rule library.

Massive data processing layer: This layer is the core of DIC platform, it can support many kinds of programming models and computing frameworks. While designing this layer, the first thing to do is ensure making many kinds of DIC framework work on the unified development platform. According to advantages and usable scope, the proper programming model can be chosen and used efficiently. This layer should let each programming model handle the calculations that they do best. It can manage and schedule uniformly resources and jobs, so as to get better performance than other computing systems which are based on the single programming model. In addition, new computing frameworks can be added to DIC platform, so massive data processing layer will support more programming models. Finally, it is necessary to use the programming model connections safely and efficiently and avoid the extra overhead to build and close connection frequently.

Massive data storage layer: While designing the massive data processing layer, it should be taken fully consideration how to store properly and access efficiently large-scale data. Structural data and unstructured data should be store and manage, respectively. This layer also provides a unified interface to access data.

Automatic data acquisition layer: The DIC platform is not only able to store and process large-scale data but also acquire new data from the external data sources constantly.

Security management layer: In a larger sense, security management layer should ensure safety of many aspects, including location of data center, power supply, host and network equipment, operation system of server, database, network connection, application system and people management. In a narrow sense, this layer includes mainly many functions related to the safety of operation system, database, application software, account management, for example, authentication, permission, system protection and safety audit.

Infrastructure layer: Using virtualization technology, infrastructure layer can transform all kinds of physical resources to resource pools represented by virtual machine, virtual network and virtual storage. This layer provides virtual resources for the upper mass data acquisition, storage and processing application on demand. It improves the reliability, customizability, manageability and scalability of DIC platform.

RESULTS AND DISCUSSION

Implementation and application: In order to verify the feasibility and effectiveness of the DIC platform architecture, a simple prototype system which was named DIC-Beginner was designed and implemented to support massive image data parallel processing. In addition, several testing applications were implemented for the morphological operation of binary image in parallel mode. Compared with serial processing mode, these applications can obtain higher speed-up ratio. In the DIC-Beginner, Hadoop and Haloop computing framework can be used, letting each kind of programming model is responsible for handling their own best computing tasks.

In order to accelerate the development of DIC-Beginner, the seven layers structure which is described in previous section was simplified: (1) Considering the system is operating in a local cluster, the security management layer is omitted, (2) Instead of "automatic data acquisition" function, experimental data are obtained in an offline way and saved on a local drive. Thus the structure of DIC-Beginner has only 5 layers. DIC-Beginner and those applications were developed and tested in Eclipse environment. Before using Hadoop and Haloop, the corresponding plugins need to be installed.

Parallel image processing algorithm based on mathematical morphology: Digital image processing technology has become widely-used methods and tools in the field of scientific computing and engineering computing. In the field of computational social science and electronic commerce it also has many successful stories. Along with the increasing scale and size, the work of managing and processing image file shows obvious characteristics of "big data". Image processing methods based on a single processor has been unable to meet the demand of user.

The traditional parallel image processing is executed mainly in multi-core or multiprocessor standalone mode. Ramraj and Rajan (2009) discussed how to make image processing algorithm be effective parallelization in a multi-core environment. In recent years, some researchers use Hadoop to solve more complex problems in image analysis and image understanding, such as image classification (Zhu, 2011) and image retrieve (Wang *et al.*, 2012b). However, researches on the low-level image processing algorithm which is implemented with Hadoop is less, especially for the parallelization of mathematical morphology algorithm.

Mathematical morphology is a kind of mature theories and methods of the digital image processing and recognition. It is composed of a set of basic operations, namely, dilation, erosion, opening operation and closing operation. The object of each operation can be either a binary image or a gray image, also can be a color image. On the basis of these operations a variety of specific and practical morphological algorithms can be combined and derived. These algorithms can be used to analyze and process the shape and structure of images, including image segmentation, image filtering, image enhancement, image restoration, boundary detection and feature extraction, etc.

Mathematical morphology uses a probe (i.e., the structural element) to collect the information of a image. When the structural element is moving in a image, we can review the relationship among the parts and understand structure characteristics of the image. Dilation, erosion, opening

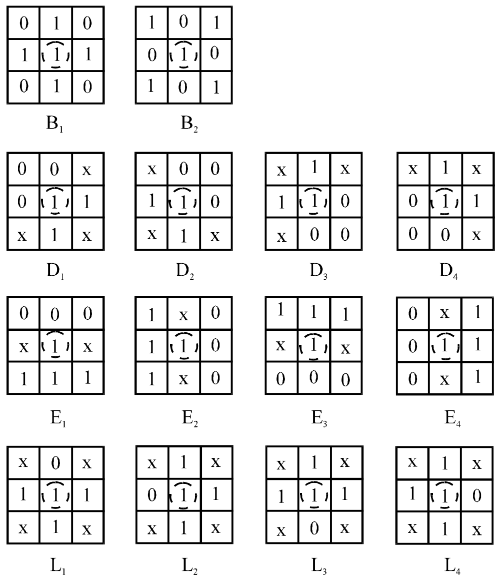

Fig. 5: Structural elements

operation and closing operation can be explained by the fact that the original image A is performed different translation (Haralick *et al.*, 1987) operation with the structural element B. Hence, mathematical morphology algorithms have natural structure for parallel implementation. Dilation, erosion, opening operation and closing operation are defined as follows:

- Dilation operation

$$A \oplus B = \bigcap \{A + b : b \in B\} \tag{1}$$

- Erosion operation

$$A \Theta B = \bigcap \{A - b : b \in B\} \tag{2}$$

- Opening operation

$$A o B = \bigcup \{B + x : B + x \subset A\} \tag{3}$$

- Closing operation

$$A \bullet B = (A \oplus B) \Theta B \tag{4}$$

In this study, dilation, erosion, opening operation and closing operation use two kinds of 3×3 structure elements (i.e., B_1 and B_2), their origin are all at the center, as shown in Fig. 5.

In addition to the above four basic morphological operation, a parallel thinning algorithm is also implemented, it is already improved. Thinning is a skeleton process in which most of image pixels are stripped continuously, eventually a wide single pixel image skeleton is gotten, without changing topology among image pixels. In the traditional thinning algorithm, structural elements $\{D_i\}$ are responsible for removing the upper left, upper right, lower right, lower left direction pixels on the four corners, structural elements $\{E_i\}$ are responsible for removing the top, bottom, left, right upward pixels. Our thinning algorithm is that adding a set of structural elements $\{L_i\}$ to strip pixels in the corner, these three kinds of structural elements (i.e., $\{D_i\}$, $\{E_i\}$ and $\{L_i\}$) will work together, as shown in Fig. 5. The dotted circle denotes the origin of structural element, "1" denotes a point on the target image, "0" indicates a point on the background image, "x" can be either a point on the target image, also can be a point of the background image.

In an iterative process, if $\{D_i\}$, $\{E_i\}$ and $\{L_i\}$ are used simultaneously to determine whether or not the outer pixels should be stripped, the convergence speed of thinning process will be greatly improved. However, not all of structural elements can be used at the same time, otherwise the connectivity of skeleton will be damaged. These structural elements are given below:

- D_i and D_{i+2}
- E_i and E_j (arbitrary $i \neq j$)
- L_i and L_j (arbitrary $i \neq j$)
- E_i and (D_{i+2} or D_{i+3})
- L_i and (D_{i+2} or D_{i+3})
- L_i and E_j (arbitrary $i \neq j$)

$$i = 1, 2, 3, 4;\ D_{i+k} = D_{(i+k) \bmod 4}$$

After the above structure elements are eliminated, the improved algorithms can be obtained:

$$S_n(X) = \bigcup_{i=1}^{4} \Psi(X,D,E,L,n,i) \qquad n = 0,1,\cdots,N \tag{5}$$

$$\begin{aligned}\Psi(X,D,E,L,n,i) = \{ &((X_n \Theta D_i) - ((X_n \Theta D_i) \circ D_i)) \cup ((X_n \Theta D_{i+1}) - ((X_n \Theta D_{i+1}) \circ D_{i+1})) \cup \\ &((X_n \Theta E_i) - ((X_n \Theta E_i) \circ E_i)) \cup ((X_n \Theta L_i) - ((X_n \Theta L_i) \circ L_i)) \}\end{aligned}$$

$$X_n = X \Theta n\ (D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus E_1 \oplus E_2 \oplus E_3 \oplus E_4 \oplus L_1 \oplus L_2 \oplus L_3 \oplus L_4) \qquad n = 0, 1, ..., N$$

As Table 1 shows, the above five kinds of binary morphology algorithm are implemented and put as test programs with DIC-Beginner.

Job management and resource management: In DIC-Beginner, Hadoop and Haloop are deployed on a cluster and they share all resources. Hadoop is used to carry out the tasks of TP1,

Table 1: Test programs

Program ID	Program name	Program function
1	TP1	Dilation
2	TP2	Erosion
3	TP3	Opening
4	TP4	Closing
5	TP5	Thinning

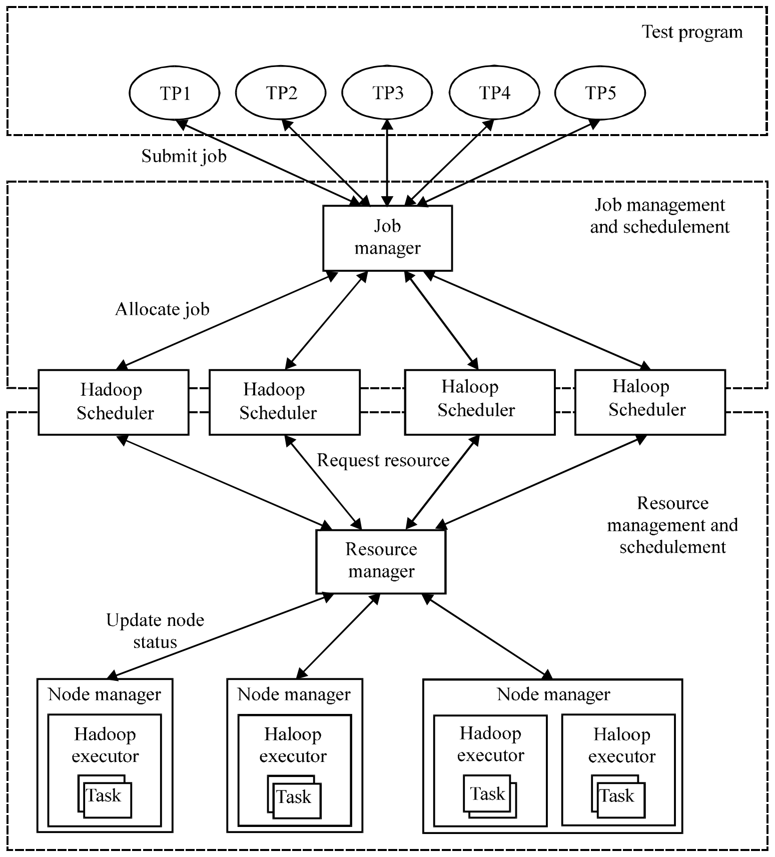

Fig. 6: Architecture of job management and resource management

TP2, TP3 and TP4. The thinning algorithm (i.e., TP5) needs to perform multiple iterative operations. MapReduce programming model of Hadoop cannot support explicitly iterative calculation. If executing TP5 with Hadoop, multiple jobs must be started and iterative process is realized in a cascading fashion. While a job is completed, an additional task must be started for judgment. The performance is poor. Thus the tasks of TP5 are performed with Haloop. Haloop extends MapReduce of Hadoop and it can support effectively iterative operation.

As Fig. 6 shows, resource manager is responsible for allocating and scheduling cluster resources, node manager runs on each node and it is responsible for providing and allocating node resources. Job manager is universally unique, it can assist developers to decompose complex jobs into subtasks and control execution flow. Hadoop scheduler and haloop scheduler take charge of not only applying for resources but also allocating resources and scheduling tasks within the slave framework. Hadoop executor and haloop executor run on the slave node, they are responsible for starting a hadoop task or haloop task.

Image file: The test data of this study comes from Stanford 40 Action Dataset. The Stanford 40 Action Dataset contains images of humans performing 40 actions. There are 9532 images in total with 180-300 images per action class. According to the resolution of image, we choose 300 images

Table 2: Test images

Group name	Resolution of images	Amount of images
A	400×300	300
B	800×600	300
C	400×300	1200

Table 3: Structural element table

	Column family	
Row key	Seinfo	History
ID	Column label	Remarks
	Sename	Name of structural element
	Rowsubscript	Row subscript of array
	Colsubscript	Column subscript of array
	Value	Value of array element

(800×600 resolution) and 1200 images (400×300 resolution). All of images had been transformed into binary images after graying and thresholding segmentation and divided into three groups as shown in Table 2.

The test image files are stored in the local file system of client machine. Before running test programs, these image files will be copied to HDFS. Finally, output data will be written to the local file system. This will make the process more efficient, because HDFS can make the best of data location, instead of the local file system. Of course, the image files also can be read directly from the local file system, then the input directory of DIC-Beginner must be specified to local file system. For TP5 program, multiple map and reduce tasks need run on the same image, so it is best to copy all image files to the HDFS so as to save bandwidth and improve efficiency.

Structural element: All structural elements (i.e., $\{D_i\}$, $\{E_i\}$, $\{L_i\}$) are stored in Hbase in the form of two-dimensional array. Table 3 shows information of the Hbase data table. For multiple records in which the values of column attribute "sename" are same, if, the line in which the value of ID is the smallest indicates the origin of a structural element.

Data flow: Test programs directly sends the path of image files to DIC-Beginner, as the input of MapReduce programs. ImageInputFormat class is responsible for generating input splits and dividing them into record. ImageInputFormat class inherits from FileInputFormat class. Analogously, ImageOutputFormat class inherits from FileOutputFormat class. It is responsible for describing the image output format of MapReduce job. In ImageOutputFormat class, ImageRecordWriter can use image file name as a key which is text type and image file content as a value which is image type, finally, store image files into HDFS.

ImageInputFormat class makes the entire image file be an input split which is a record. For a key/value pair, the key is the text type which stores the image file path in the file system, the value is a record which is imageWritable type. ImageWritable class rewrites two methods of the Writable interface. These methods write and read data content of images, respectively. Input split is the input block of map operation, each map operation only deals with one input split but this split is not the image file itself, it is a reference to the data. The input file will be always divided into several units by default. In order to make map task treat the whole image file as one record, ImageInputFormat defines two methods:

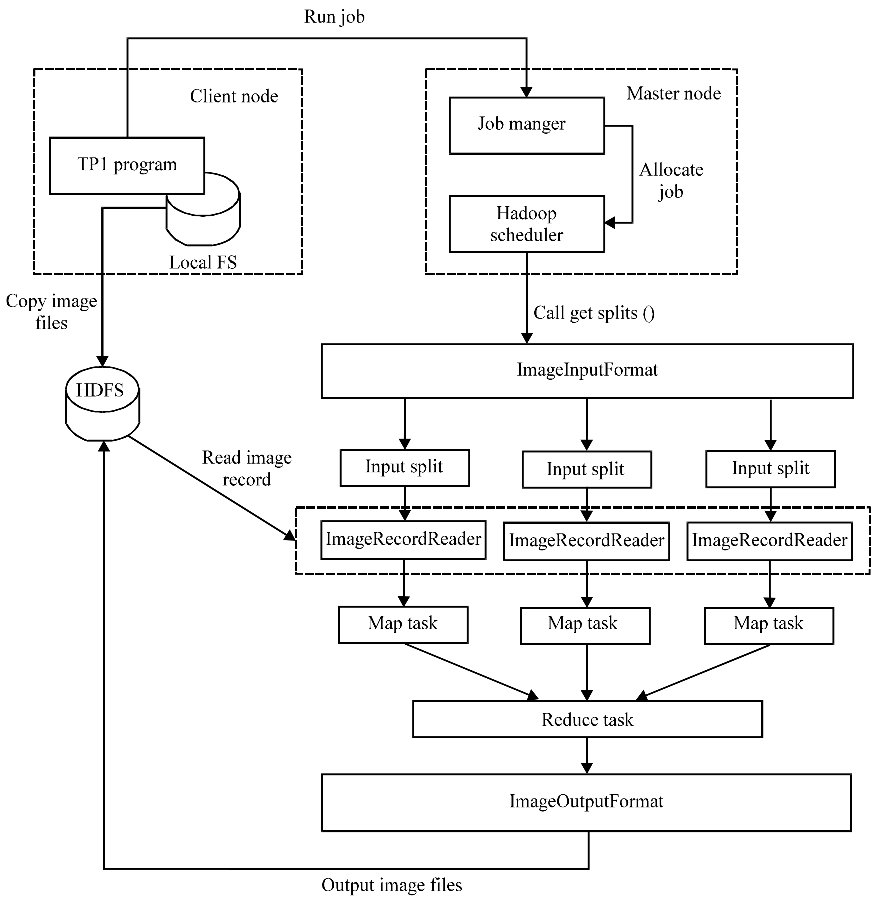

Fig. 7: Data flow of TP1

- Overriding is Splitable ( ) method and set return value as false
- Overriding GetRecordReader ( ) method and returning a custom ImageRecordReader to read the entire image file

Figure 7 shows the data flow of TP1 program with Hadoop. When TP1 starts on DIC-Beginner, JobManager calls getSplits ( ) method of ImageInputFormat class. Assigning the sum of the image file to numSplits, the desired number of map tasks is determined. After creating input splits, the client machine will send them to the HadoopScheduler. According to the information about storage location in the input splits, HadoopScheduler will schedule them to the HadoopExecutor.

On HadoopExecutor, map task will transfer input split to GetRecordReader ( ) method of ImageInputFormat, so as to obtain ImageRecordReader. Map tasks use ImageRecordReader to read the record and generate key/value pairs which will be passed to map functions.

Infrastructure setup: For our experiments, the infrastructure of DIC-Beginner consists of local cluster of 5 machines: 1 of the machines is both master node and client node, the other 4 machines are converted into 8 slave nodes using VMware virtualization technology. Each machine has 1 quad-core Intel i7-3770 (3.40 GHz) processor with 16 GB memory. All machines are connected via a megabit Ethernet switch.

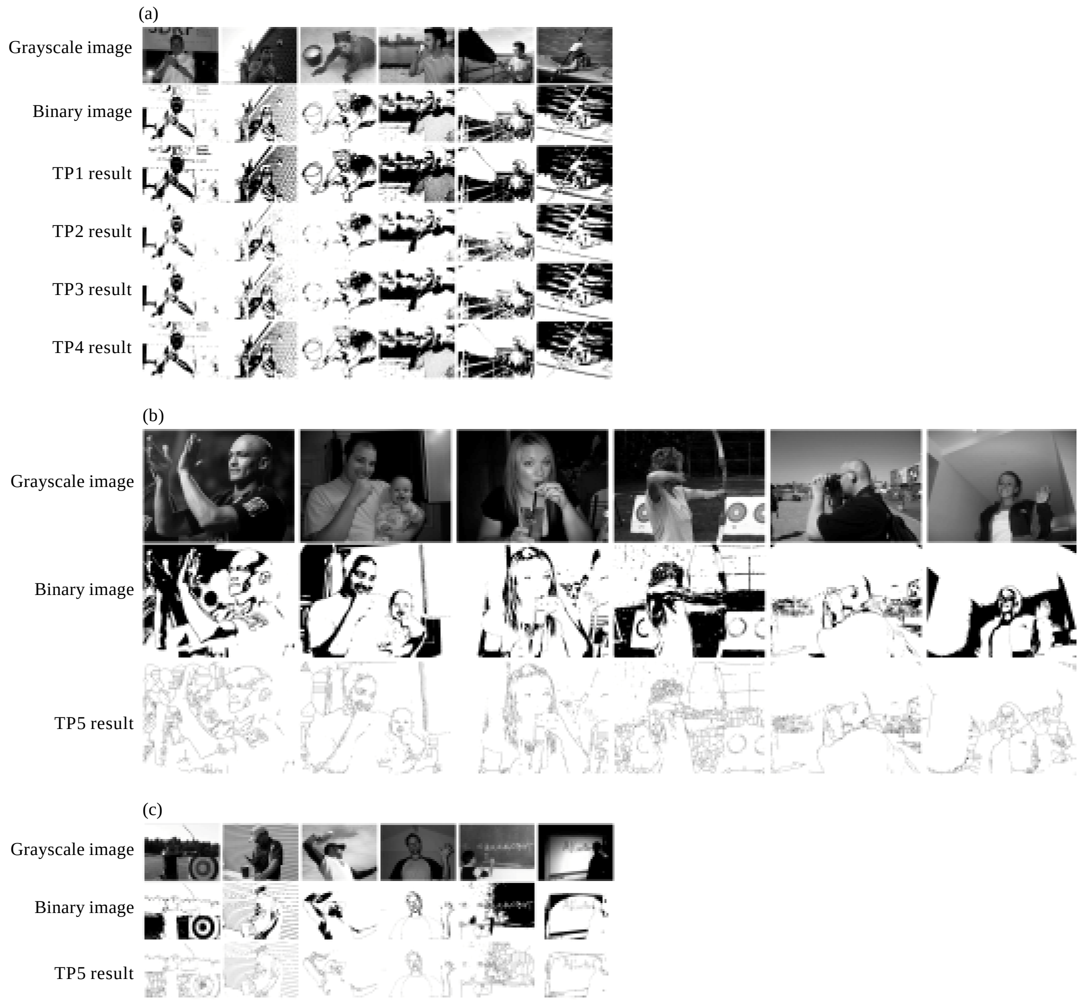

Fig. 8(a-c): Part of test data and experimental results (a) Group A, (b) Group B and (c) Group C

Result and evaluation: In the beginning, using the first group of test data (i.e., Group A) in Table 2 as input, the first four test programs were carried out. Then TP5 program was run for the other two groups of test data in Table 2. Figure 8 shows part of test data and experimental results. In all experiments, there were two workers per slave node and each worker was pinned to use one of two cores.

Figure 9a shows application speedup of DIC-Beginner has a satisfactory scaling performance as the number of workers increases from 2 to 16 for the group A of test data. However, as the number of workers increases to 16, DIC-Beginner's overhead is no longer negligible relative to applications' own computation, resulting in 20% less than ideal speedup.

Figure 9b shows relative runtime of TP5 as the number of workers increases from 2 to 16 for group B and group C which have similar amounts of data. TP5-B and TP5-C correspond to results of TP5 on both groups of test data, respectively. Because the size of files is far less than the size of HDFS block and there are many files, each map operation will only deal with few input data. As a result, there will be a lot of map tasks, each new map operation can cause some performance loss.

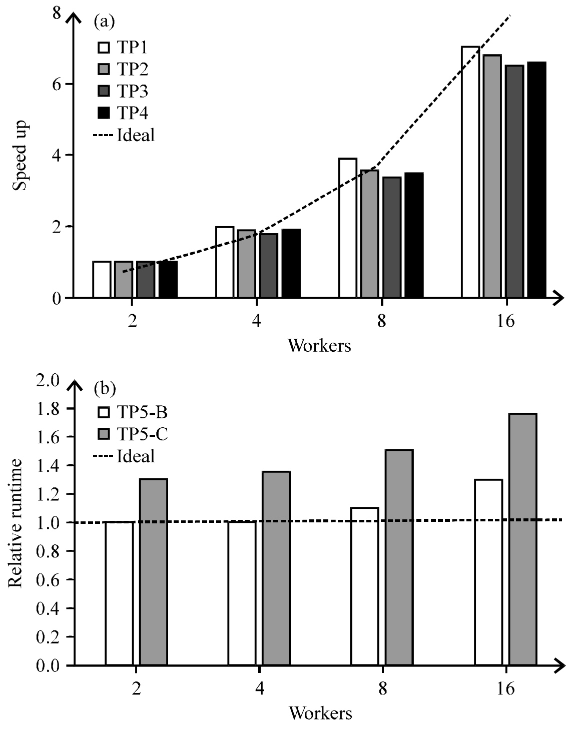

Fig. 9(a-b): Scaling performance, (a) Speed up and (b) Relative runtime

CONCLUSION

In this study, an architecture of DIC platform with seven layers is studied, the requirements of designing DIC platform are analyzed and the DIC integrated research method is discussed. Finally, as a prototype system, DIC-Beginner was implemented to support mass image data parallel processing, This study has high theoretical significance and application value.

REFERENCES

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph and R.H. Katz *et al.*, 2009. Above the clouds: A Berkeley view of cloud computing. Technical Report No. UCB/EECS-2009-28, Department of Electrical Engineering and Computer Science, University of California, Berkeley, February 10, 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

Borkar, V., M. Carey, R. Grover, N. Onose and R. Vernica, 2011. Hyracks: A flexible and extensible foundation for data-intensive computing. Proceedings of the IEEE 27th International Conference on Data Engineering, April 11-16, 2011, Hannover, pp: 1151-1162.

Chen, G.L., G.Z. Sun, Y. Xu and M. Lu, 2008. Methodology of research on parallel algorithms. J. Chin. Comput., 31: 1493-1502.

Chen, G.L., G. Sun, Y. Xu and B. Long, 2009. Integrated research of parallel computing: Status and future. Chin. Sci. Bull., 54: 1845-1853.

Chen, C.L.P. and C.Y. Zhang, 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Inform. Sci., 275: 314-347.

Foster, I., C. Kesselman and S. Tuecke, 2001. The anatomy of the grid: Enabling scalable virtual organizations. Int. J. High Perform. Comput. Applic., 15: 200-222.

Foster, I., C. Kesselman, J.M. Nick and S. Tuecke, 2003. The Physiology of the Grid. In: Grid Computing: Making the Global Infrastructure a Reality, Berman, F. (Ed.). John Wiley and Sons, New York, pp: 217-249.

Garber, L., 2012. Using in-memory analytics to quickly crunch big data. Computer, 45: 16-18.

Guo, M. and C. Xu, 2011. Programming model and separative resource management for cloud computing. Commun. CCF, 7: 26-33.

Haralick, R.M., S.R. Sternberg and X. Zhuang, 1987. Image analysis using mathematical morphology. IEEE Trans. Patt. Anal. Mach. Intell., 9: 532-550.

Hayes, B., 2008. Cloud computing. Commun. ACM., 51: 9-11.

Howe, D., M. Costanzo, P. Fey, T. Gojobori and L. Hannick *et al.*, 2008. Big data: The future of biocuration. Nature, 455: 47-50.

Kourtesis, D., J.M. Alvarez-Rodriguez and I. Paraskakis, 2014. Semantic-based QoS management in cloud systems: Current status and future challenges. Future Gener. Comput. Syst., 32: 307-323.

Luo, J.Z., J.H. Jin, A.B. Song and F. Dong, 2011. Cloud computing: Architecture and key technologies. J. Commun., 32: 3-21 (In Chinese).

Marinos, A. and G. Briscoe, 2009. Community cloud computing. Proceedings of the 1st International Conference on Cloud Computing, December 1-4, 2009, Beijing, China, pp: 472-484.

Marz, N. and J. Warren, 2013. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. O'Reilly Media, US.

Ramraj, E. and A.S. Rajan, 2009. Using multi-core processor to support network parallel image processing applications. Proceedings of the International Conference on Signal Processing Systems, May 15-17, 2009, Singapore, pp: 232-235.

Wang, J., D. Crawl and I. Altintas, 2012a. A framework for distributed data-parallel execution in the Kepler scientific workflow system. Procedia Comput. Sci., 9: 1620-1629.

Wang, X.W., Q.Y. Dai, W.C. Jiang and J.Z. Cao, 2012b. Retrieval of design patent images based on mapreduce model. J. Chin. Comput. Syst., 33: 626-632.

Winter, R., 2008. Why are data warehouses growing so fast? April 10, 2008. http://searchdatamanagement.techtarget.com/news/2240111227/Why-Are-Data-Warehouses-Growing-So-Fast.

Zhu, Y.M., 2011. Image classification based on hadoop platform. J. Southwest Univ. Sci. Technol., 26: 70-73.