Presented at the Asian Technology Information Program (ATIP) 3rd China HPC Workshop
SC07 Conference, Reno, Nevada, November 11, 2007

The Design Methodology of Phoenix Cluster System Software Stack

Jianfeng Zhan, Lei Wang, Bibo Tu, Hui Wang, Zhihong Zhang, Yi Jin, Yu Wen, Yuansheng Chen, Peng Wang, Bizhu Qiu, Dan Meng, Ninghui Sun

Key Laboratory of Computer System and Architecture
Institute of Computing Technology
Chinese Academy of Sciences

jfzhan@ncic.ac.cn

ATIP 3rd China HPC Workshop, November 11, 2007, Reno, NV

The Design Methodology of Phoenix Cluster System Software Stack

Jianfeng Zhan, Lei Wang, Bibo Tu, Hui Wang, Zhihong Zhang, Yi Jin, Yu Wen, Yuansheng Chen, Peng Wang, Bizhu Qiu, Dan Meng, Ninghui Sun

Key Laboratory of Computer System and Architecture

Institute of Computing Technology
Chinese Academy of Sciences, Beijing 100080, China

jfzhan@ncic.ac.cn

ABSTRACT

Though many research groups have explored the design methodology of cluster system software stack, few works discuss what constitutes a good one. In this paper, we choose four criteria throughout the lifecycle of cluster system software stack to evaluate its design methodology, including code reusability, evolveability, adaptability and manageability. According to the four criteria, we have proposed a management service-based layered design methodology and built a complete cluster system software stack for both scientific and business computing. Our practices and evaluations show our design methodology has advantages over others in terms of the proposed criteria.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design – *distributed systems*.

General Terms

Management, Design, Measurement.

Keywords

Cluster system software stack, Design Methodology, Evaluation Criteria, Practice

1. INTRODUCTION

Though many research groups [1-6] have explored the design methodology of cluster system software stack, few works discuss what constitutes a good one. The design methodologies have been investigated without explicit consensus on evaluation criteria [1-6], most of which are hidden in the practice. Since the application range of cluster has expanded beyond scientific computing and become dominant production platforms for different kinds of computing, it is worth proposing evaluation criteria and exploring new design methodology for cluster system software stack[i].

In some projects, monolithic cluster system software stacks are developed from scratch for different contexts (scientific or business) [7-8] or scenario (search engine) [9-11] case by case. In this paper, we call it building-from-scratch methodology. This methodology provides little support for code reuse[ii].

Other works [1] just package, or integrate the best practice, which we call packaging-or-integrating methodology. This methodology lowers the manageability of cluster system software, since varieties of daemons with redundant functions provide inconsistent information and redundancy of software modules makes deciding their running states or troubleshooting [12] much harder..

Focusing on scientific computing, SSS project [2-4] and DOE CCA project [5-6] resort to component [29] technology to support code reuse and component substitution, which we call component-centered methodology. These related works are major steps to push forward the design methodology of cluster system software, but they provide no systematic evaluation criteria. Besides, they pay no attention to the fact that the application range of cluster has expanded beyond scientific computing and fail to provide common component framework for different computing contexts. For example detection and process management modules are required for system monitoring [13] [14], job management system [7] for scientific computing and web hosting environment [8] for business computing. Under this context, we need coarser granularity of code reuse than component.

In order to evaluate the existing design methodologies and explore new one, we choose four evaluation criteria throughout the lifecycle of cluster system software stack. Among them, three criteria are qualities of cluster system software stack: evolveability, adaptability and manageability, which reflect the superiority of design methodology in terms of maintenance or development cost, and the other criterion-code reusability reflects the productivity of the design methodology itself. According to those evaluation criteria, we have proposed and evaluated a new layered design methodology. Our practices and qualitative evaluation show this new design methodology has advantage over others in terms of the proposed criteria.

To the best of our knowledge, this is the first paper to propose the evaluation criteria for the design methodology of cluster system software stack. The research contribution of this paper can be concluded as:

[i] This work is supported by the National 863 High-Tech Program of China (Fully integrated system software).

[ii] This is an extended version of IEEE Cluster 2007 poster: a layered design methodology of Cluster system stack.

a) We bring forward four explicit criteria to evaluate the design methodology of cluster system software stack, including code reusability, evolveability, adaptability and manageability.

b) We choose the management service as basic building block and propose a management service-based layered design methodology to build cluster system software stack with different layer concentrating on different functions. Furthermore, we extract common sets of core management service as reusing framework for different computing context.

c) We have evaluated four main design methodologies in term of the proposed four evaluation criteria. The evaluations show our design methodology has advantage over others.

This paper is structured as follows: Section 2 outlines related works; section 3 proposes and discusses the evaluation criteria of the design methodologies; section 4 characterizes the new design methodology; Section 5 summarizes the practices of building Phoenix system; Section 6 evaluates the design methodologies in terms of the proposed evaluation criteria; finally, in section 7, we give a conclusion.

2. RELATED WORKS

The usage model plays vital important role in designing system software for high performance computing. Two main classes of usage models are classically identified for high-end computing systems: capability-oriented and capacity-oriented usage. Capacity-oriented cluster system software stack provides more complex services and supports more flexible models such as timesharing than capability-oriented one [17]. In this section, we summarize different design methodologies of cluster system software stack for different usage models.

2.1. The Design Methodologies of Capability-oriented System Software Stack

Capability-oriented system software stacks aim at supporting high-end scientific computing, most of which adopt KISS (keep it simple and stupid) principle.

In designing the BG/L system software, IBM[15][16] followed three major principles (simplicity, performance, familiarity) and adopted strictly space-sharing mode, which means that only one (parallel) job can run at a time on a BG/L partition. All simplifications targeted at improving the performance and reliability of system, and IBM leveraged existing components while implementing critical components from a clean slate.

Paper [17] presents a component-based approach for developing dedicated high-end computing operating systems. On the base of Fractal generic component model, this research group enhances and adapts Fractal to the specific requirements of HEC, which enables the construction of dedicated OSs according to application needs, hardware characteristic, and the associated programming model.

2.2. The Design Methodologies of Capacity-oriented Cluster system software stack

The cluster system has become the dominant production platform for both scientific and business computing. Under this context, for most of computing scenarios, users prefer to capacity-oriented use model.

With capacity-oriented usage model, many research groups have developed several monolithic cluster system software stacks for different contexts (scientific or business computing) [7-8] or scenarios (search engine) [9-11] from scratch without code reuse, which we call building-from-scratch methodology.

Several research groups follow packaging or integrating methodology, such as OSCAR project [1], SCE [18] from Thailand Kasetsart University, Score [19] from Japan Real world computing project. Among them, the typical system is OSCAR, taking the best of what is currently available and integrating it into one package. This methodology results in bad manageability, since varieties of daemons with redundant functions provide inconsistent information and redundancy of software modules makes deciding their running states or troubleshooting [12] much harder.

Focusing on scientific computing, several projects resort to component [20] to promote code reuse and component substitution, which we call component-centered methodology. SSS project [2-4] is to address the lack of software for the effective management of terascale computational resources and develop an integrated suite of machine independent, scalable systems software components for scientific computing; the rational behind DOE CCA project [5-6] is to deal with the complexity of developing interdisciplinary HPC application through introducing higher level abstractions and allowing code reusability. These related works are major steps to push forward the design methodology, but they provide no evaluation criteria and pay no attention to the fact that the application range of cluster has expanded beyond scientific computing. These works fail to provide common component framework for different computing contexts.

2.3. Evaluation Criteria of Cluster system software stack

Few works discuss the evaluation criteria for design methodology of cluster system software stack. Papers [2-6] notice the importance of code reuse and system evolveablity for evaluating design methodologies, but they fail to provide systematic evaluation criterion.

3. EVALUATION CRITERIA OF METHODLOGY

3.1. The Rational for Choosing Evaluation Criteria

When choosing sound criteria for evaluating design methodology, we take them into accounts facts and rule of thumb found in software engineering and cluster practices [20] [21], and discuss each evaluation criterion under these context.

Throughout the lifecycle of cluster system software stack, we choose four criteria to evaluate and explore design methodologies. Among them, three criteria are qualities of cluster system software stack: evolveability, adaptability and manageability, which reflect superiority of design methodology in terms of maintenance or development cost, and the other criterion-code reusability reflects the productivity of the design methodology itself.

3.1.1. The Code Reusability

Code reusability is varying for different design methodologies. The code reusability greatly contributes to the productivity and robustness of cluster system software. High reusability means

higher productivity and robustness, so it is a key criterion for evaluating productivity of design methodology.

3.1.2. The evolveablity of cluster system software stack

In the lifecycle of cluster system software stack, we need update or substitute modules. The evolveablity of cluster system software stack reflects the quality of the design methodology of cluster system software stack in terms of maintenance cost.

3.1.3. The adaptability of cluster system software stack

The application range of cluster has expanded beyond scientific computing and become mainstream production platforms for different computing contexts. The adaptability of cluster system software stack to different kinds of computing is an important criterion for evaluating design methodology.

This could be measured in terms of two dimensions:

(a) The extent to which cluster system software stack for different computing context share the same code base.

(b) The application range for which cluster system software stack is suitable, e.g. scientific or business computing.

3.1.4. The manageability of cluster system software stack

The manageability of cluster system software stack has big impact on the maintenance cost of cluster system, which takes a lot of share of total cost of ownership (TCO) of cluster system.

The design methodology has a great impact on manageability of cluster system software stack. For example packaging methodology results in bad manageability, since varieties of daemons with redundant functions may provide inconsistent information. The manageability of cluster system software stack reflects the quality of design methodology in terms of maintenance cost of cluster system, so we choose it as an evaluation criterion.

3.2. The Discussion of Evaluation Criterion

It is worth pointing out that some users prefer to specific perspective in choosing the design methodology of cluster system software stack. For example, for capability-oriented system software stack, the system designer prefers to dedicated or strictly space-sharing use mode, he will ignore the evaluation criterion such as adaptability of cluster system software stack. But in this paper, we take it into account the general case, so our evaluation criteria still hold true for most cases.

4. THE DESIGN METHODOLOGY OF PHOENIX

According to those evaluation criteria, we propose a management service-based layered design methodology to build Phoenix cluster system software stack. The new design methodology includes three choice points.

4.1. Management Service-Based

Management service is defined as basic entity in cluster software infrastructure with open and documented application programming interface (API), which provides core functions for different cluster system subsystems, such as system monitoring tool, job management tool, or web hosting environment. Figure 1 is some represented API of parallel processes management service.

Execute a command on single hosts:

int exec_single_host(string host,string user, string cmdline, int nmili_sec, int fexitcode, exec_result& eresult);

Execute a command serially on several hosts:

void seri_exec_cmd(vector<string>& hosts, string user, string cmdline, int nmili_sec, int fexitcode, vector<exec_result>& eresults);

Simultaneously execute a command on several hosts:

int para_exec_cmd(vector<string>& hosts, string user, string cmdline, int nmili_sec, int fexitcode, vector<exec_result>& eresults, unsigned grp_num = 16);

Figure 1. Some represented API

Management service is chosen as basic building block, and core functions are distributed on different management services. This decision point is to improve the evolveablity of cluster system software stack, since we could update or substitute service, keeping on same semantic and syntactic of service access interface.

4.2. Layered Design

We choose a layered architecture style to build cluster system software with different layer concentrating on different functions, and divide Phoenix cluster system software stack into four layers: resource, Phoenix Engine, runtime environment and Phoenix Portal as shown in Figure 2;

Mary Shaw [21] points out several desirable properties of layered architecture style. First, it supports designs based on increasing levels of abstraction. Second, it supports enhancement, and changing the function of one layer will not affect at most two other layers. Third, it supports reuse.

Besides these, layered architecture style helps to hide complexities of lower layer to higher layer. E.g. user interacts with runtime environment via Phoenix Portal, and the complexities of Phoenix Engine are transparent to users.

4.3. Reusing Service Framework

Choosing a complete service framework as reusing unit for different computing context is to improve the code reusability, control the consumption of human resource and improve the adaptability of Phoenix cluster system software stack.

We extract common sets of core services as reusing framework for both scientific and business computing, which is named Phoenix Engine. We build Phoenix Engine with layered architecture. Scalable fault-tolerance support is implemented as core management service, embedded within Phoenix Engine, and the complexity is transparent to the developers. This decision point is to decrease the difficulty level of developing new cluster system software.

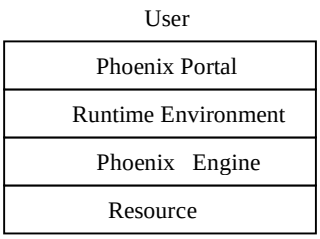

Figure 2. Layered architecture of Phoenix

5. THE PRACTICE OF BUILDING PHOENIX

In section 5.1, we describe the architecture of Phoenix system. Then, we introduce the practice of building Phoenix Engine and different runtime environments on Phoenix Engine respectively in section 5.2 and 5.3.

5.1 The Architecture of Phoenix System

Fig.2 describes the layered architecture of Phoenix system. The lowest layer is resource. The third layer is Phoenix Engine, which defines common sets of core services, including scalability and fault-tolerance support. The second layer is runtime environment, through which users utilize cluster resources to fulfill their targets. The First Layer is Phoenix portal, the GUI for different user roles. In Phoenix, we define four user roles, including system constructor, system administrator, scientific computing users, and business computing user, for whom Phoenix have provided difference runtime environments as follows:

a) System constructor configures, deploys and boots cluster system with system construction tool [22], which is similar to Rock [23].

b) System management and monitoring tools [14] assist system administrators to perform daily system management, real-time system monitoring, performance and fault analysis.

c) Job management system [23] is a runtime environment for managing cluster resources, through which scientific computing users submit their jobs and complete their computing task.

d) Business hosting environment manages multi-tier business applications and guarantees their high-availability and load-balancing.

5.2. The Practice of Building Phoenix Engine

5.2.1. Phoenix Engine- a Reusing Service Framework

Phoenix Engine is the common set of core management services with published API in several forms (Such as Socket, RPC and ORB etc). The architecture of Phoenix Engine is shown in Fig. 3.

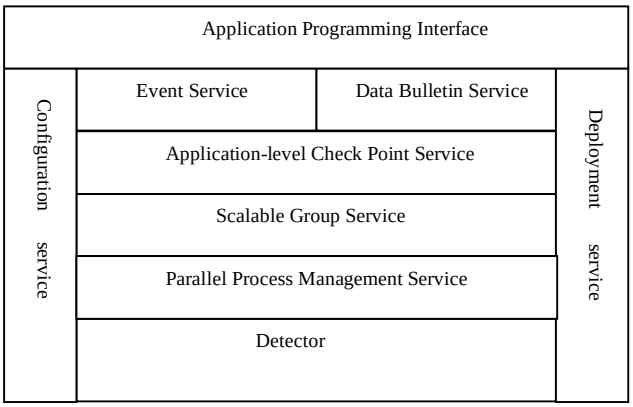

Figure 3. Architecture of Phoenix engine

5.2.1.1. Configuration Service

It provides cluster-wide configuration information, and supports dynamic reconfiguration. Configuration service has a self-introspection mechanism to automatically find cluster resources.

5.2.1.2. Deployment Services

They are collection of important services for deploying operating system, Phoenix Engine and Runtime environment, including DHCP, TFTP and FTP etc.

5.2.1.3. Detector

Detector service includes resource detector and application state detector. Resource detector monitors usage of physical resources, such as CPU, memory, swap, disk I/O and network I/O of nodes, switch and storage, which are fundamental for many runtime environments; application state detector monitors application status such as physical resources consumed by specific application, the application healthy status, as well as application information related to system level agreement.

5.2.1.4. Parallel Processes Management Service

It supports remote job or service loading, killing, and resource cleaning.

5.2.1.5. Scalable Group Service

Scalable group service is to solve scalability and fault-tolerance simultaneously. Its key functions include monitoring states of nodes and networks; creating its own meta-group with high availability guarantee; providing interfaces for creating, joining, leaving and guaranteeing their high availability of upper-layer service groups.

5.2.1.6. Application-level Checkpoint Service

Based on scalable group service, it provides interfaces for upper-layer services to save state data, and upper-layer services themselves are responsible for saving and deleting state by calling interfaces of application-level checkpoint service.

5.2.1.7. Event Service

Based on scalable group service, event service plays the role of communication channel, and provides following interfaces: registering event supplier and its produced event types; registering event consumer and its interested event types; providing events filtering and event notification.

5.2.1.8. Data Bulletin Service

Based on scalable group service, data bulletin service is an in-memory data management subsystem, which provides interfaces for data submission, non-persistent data storage and query. Detectors store states of cluster-wide physical resource and application state in the data bulletin service for queries from different runtime environments.

5.2.2. Collaborations among management services

To reserve space, we give a short introduction of collaborations among management services.

With Phoenix stack, the whole cluster system is divided into several partitions, and each partition is composed of one server node, at least one backup node, and other computing nodes. Different from the partition of Blue Gene/L, we support time-sharing mode, and different jobs or services could run simultaneously in the same partition. In order to achieve scalability, each partition chooses one server node as representative, and they form a meta-group, as shown in Figure 4. Among different partitions, the principal part of group management is SGS (scalable group service). A SGS takes charge of a partition. The SGSes form a meta-group with a ring structure, and complies with a light-weight membership protocol.

Data bulletin service (DB, as shown in Figure 4), checkpoint service, and event service call the interface of scalable group service to create service group and register policies on how to deal with failures. The fault-tolerance support provided by SGS is transparent to the caller of those Service. Taking data bulletin service (DB) as an example, if one member of DB service group fails, SGS on the same host will notify all members of SGS group and then restart the failed service. If the host node of one data bulletin service fails, SGS member on the next host next to the failed one in the ring structure will select a new node for migrating SGS and then recover that data bulletin service.

Because of the page limit, the detail of scalability and fault-tolerance support can be found in paper [24].

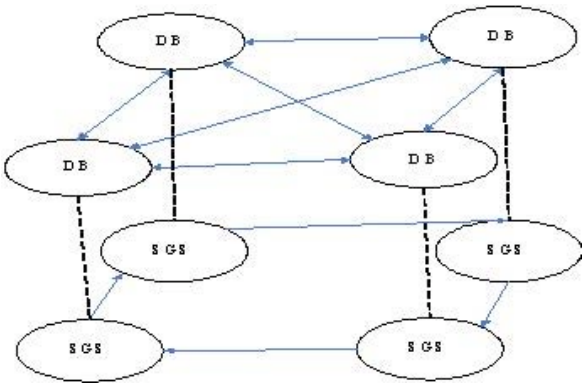

Figure 4. Collaborations among management services framework

5.3 The Practices of Building Runtime environments on Phoenix Engine

On the Phoenix Engine, we have built five different runtime environments, including Builder- a system construction tool[20], PWS-a job management system[24][25], BRE-a runtime environment for business computing, GridView-a system monitoring tool[13], Admin-a system administration tool, most of which have been deployed on production systems. These systems have been verified and deployed on Dawning 4000A super server [26] for scientific computing and other cluster systems for business computing.

5.3.1 Runtime Environment for business computing

In this section, a runtime environment for digital library named Phoenix-BRE is used to discuss how to construct new runtime environment on the basis of Phoenix Engine, which provisions resource for distributed application of digital library, such as information retrieval.

In order to achieve the performance goals of users, we present the following solution: the index data is loaded into memory after the system startup, and several server nodes with big memory form a server group running an information retrieval instance. The architecture of an information retrieval instance is illustrated in Figure 5. The front end load balancer dispatches requests, while the system (resource provision server) dynamically adjust (increase or decrease) the number of instances of information retrieval according to the load status of different server groups. When the load is going down, Phoenix-BRE will make some nodes sleep; and when the load is coming up, Phoenix-BRE will increase the number of instances of server groups and bring up more instances of information retrieval online.

Besides these functions mentioned above, Phoenix-BRE also provides other distributed application management features such as dynamic deployment, remote start/shutdown, and guarantee of quality of service. Figure.6 is the architecture of Phoenix-BRE on the base of Phoenix Engine.

With Phoenix Engine, Phoenix-BRE has several desirable properties as follows:

(1) Phoenix Engine provides most functions of Phoenix-BRE, and the effort of developing Phoenix-BRE just spends on the user interface and other module such as resource provision server.

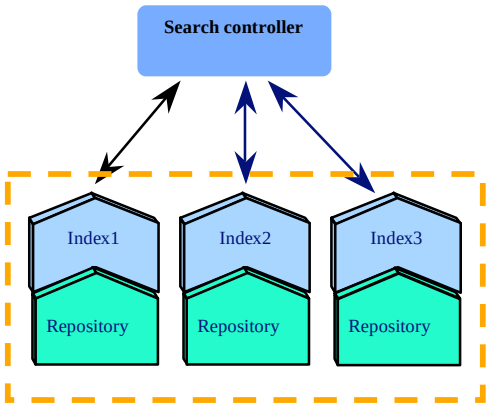

Figure 5. An information retrieval system instance

(2) The scalability of Phoenix-BRE system is improved on the basis of management service groups. Detectors export states of physical resources or applications to data bulletin service group, while resource provision server obtains cluster-wide information directly from data bulletin service group, gets real-time notification of failure or performance event from event service group, and makes decision of increasing or decreasing the number of application (information retrieval) instances according to these information.

(3) The fault-tolerance of Phoenix-BRE system is guaranteed with the support of scalable group service. The failed service,

e.g. data bulletin service, will be restarted and come to work in a short period of time.

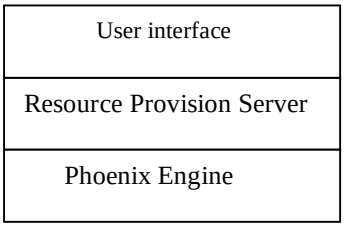

Figure 6. Architecture of Phoenix-BRE

In Figure 7, the screen snapshot shows the number of instances of digital library application is increasing with the load increases. The red color indicates the CPU load.

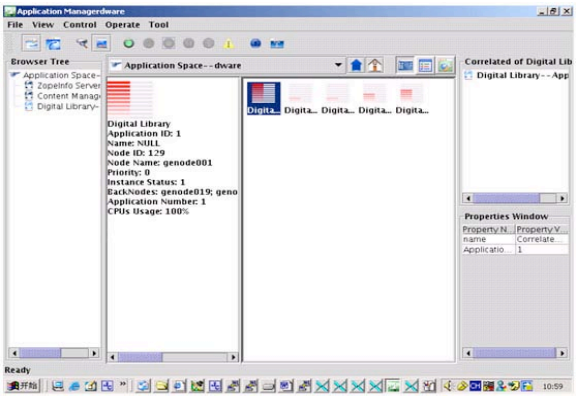

Figure 7. GUI of Phoenix-BRE.

5.3.2 Review of Building Runtime Environment for scientific computing

In paper [24] [25], we have discuss how to construct Phoenix-PWS (Partitioned Workload Solution), a job management system on the base of Phoenix Engine. PWS is an improved and modified version of OpenPBS (Portable Batch System) [7]. PWS supports multi-pools with customized scheduling policies for different pools and dynamic leasing among different pools.

Comparing with the OpenPBS, PWS has several desirable properties as follows:

(1) Phoenix Engine provides most functions of PBS, and developing new PWS system focuses on the user interface and scheduling modules and server.

(2) The scalability of PWS system is improved on the basis of service groups such as data bulletin. All physical resources detectors export resource information and job status information to data bulletin service group, from which the PWS system directly obtains cluster-wide information, thus the system workload is reduced and system scalability is improved; By registering events of node, network and application failure to event service group, PWS can get real-time notification of those events, while PBS needs polling periodically and consumes network bandwidth.

(3) The fault-tolerance of PWS system is guaranteed on the base of scalable group service. If one data bulletin service fails, the state information provided by other data bulletin services can still be obtained. With the support of SGS, the failed data bulletin will be restarted and come to work in a short period of time.

(4)The scheduler group for different pools is created on the basis of scalable group service with high availability guaranteed, while PBS doesn't guarantee it.

(5) PWS supports multi-pools and dynamic leasing among different pools.

6 Evaluation of Design Methodologies

In this section, we will evaluate four main design methodologies of capacity-oriented cluster system software in terms of the proposed evaluation criteria. Since capability-oriented system stack only focuses on high-end scientific computing and adopts dedicated usage model, we do not compare its design methodology with other main methodologies.

In this section, we analyze the pros and cons of four main design methodologies in terms of four evaluation criteria. Since it is hard to quantitatively evaluate design methodology, we give a qualitative analysis of each design methodology. The evaluation results are listed in Table 1.

6.1. Building-from-scratch Methodology

- The code reusability is low, since it provides little supports for code reuse.
- The evolveability is low, since most of the system built with this methodology is monolithic, hard to update or substitute specified modules.
- The adaptability of cluster system stack is low, since you need build system for different computing context from scratch.
- In terms of manageability of cluster system stack, it is just same as the package-or-integrating methodology, since most of users have to integrate or package different system built from scratch by different group, e.g. Ganglia [13] for system monitoring and Rock [23] for system building.

6.2. Package-or-integrating Methodology

- The reusability is high, since it reuses the whole system, e.g. Ganglia for system monitoring.
- The evolveability is low, since most of integrated package are monolithic, hard to update or substitute.
- The adaptability of cluster system stack depends upon that of existing packages, most of which focus on system monitoring, job management system, system administration and system building. For the average case, the flexibility is middle.
- The manageability of cluster system stack is low, since software systems from different groups, integrated into one package without interoperation, may provide inconsistent information, and redundancy of software modules makes deciding their running states much harder.

6.3. Component-centered methodology

- The code reusability is middle, since reuse unit is component of small granularity. But the reusability may be improved with well-designed component framework.
- The evolveability is high, since component is chosen as basic unit for update or substitution.
- The adaptability of being suitable to different kinds of computing context may be improved, but existing related

works just componentalize cluster system software for scientific computing.

- The manageability of cluster system stack is high, since component-based system eliminates the redundancy of software modules, which is found in packaging-or-integrating methodology, and provides consistent information.

6.4. Our Management Service-based Layered Design Methodology

- The reusability is high, since we choose a complete service framework-Phoenix Engine as reusing unit.
- The evolveability is high, since service is chosen as basic unit for update or substitution.
- The adaptability of being suitable to different kinds of computing is high, because we extract common sets of core service for different computing contexts. In practice, we have constructed user environments sharing same code base for both scientific and business computing.
- The manageability of cluster system stack is high, since functions are distributed on different services and they collaborate to provide consistent information.

6.5. Summary of Evaluations

The evaluation results are summarized in TABLE 1. For each criterion, methodologies scored high are labeled with asterisk (*), while methodologies scored low are underlined. The number of each methodology is listed in first row. Among them, the No.1 is building-from scratch methodology; the No.2 is packaging-or-integrating methodology; the No.3 is component-centered methodology. The No.4 is our management service-based layered design methodology.

Table 1. Evaluation scores of design methodologies

Criteria	Methodology			
	No.1	No.2	No.3	No.4
reusability	3	5*	4	5*
manageability	3	3	5*	5*
evolveablity	3	3	5*	5*
adatability	3	4	4	5*
sum	12	15	18	20

For each evaluation criterion, the rule for assigning score is as follows:

- Higher code reusability means higher score.
- Higher evolveability of cluster system software stack means higher score.
- Higher flexibility of cluster system software stack means higher score.
- Higher manageability of cluster system software stack means higher score.

The evaluation results are summarized in Table 1. For each criterion, methodologies scored high are labeled with asterisk (*), while methodologies scored low are underlined. The number of methodology is listed in first row.

In terms of the proposed criteria, the No.1-building-from-scratch methodology gets no high scores and four low scores in terms of all four criteria.

The No.2: packaging-or-integrating methodology gets one high scores: (1) high reusability. The No.2 methodology gets two low scores: (1) low manageability and (2) low evolveablity.

The No.3: component-centered methodology gets two high scores: (1) high manageability and (2) high evolveablity.

The No.4: our layered methodology gets four high scores: (1) high reusability, (2) high manageability, (3) high evolveablity and (4) high flexibility.

TABLE 1shows the No.4: our management service-based layered design methodology has advantage over other three methodologies in terms of all four evaluation criteria.

7. Conclusion

Though many research groups have explored design methodology of building cluster system software stack, few works discuss what constitute a good one.

In this paper, we propose four criteria throughout the lifecycle of cluster system software stacks to evaluate different design methodologies. Among them, three criteria are qualities of cluster system software stack: evolveability, adaptability and manageability, which reflect the superiority of design methodology in terms of maintenance or development cost, and the other criterion-code reusability reflects the productivity of the design methodology itself.

We propose a management-service based layered design methodology to build cluster system software with different layer concentrating on different sets of functions. This design methodology has three decision points: (1) management service-based; (2) layered design and (3) reusing service framework. Following this design methodology, we have built a complete capacity-oriented cluster system software stack for both scientific and business computing. On the basis of Phoenix Engine, we have constructed more complete runtime environments with ordinary efforts, including system construction tool, system management tool, system monitoring tool, job management system, and business hosting environment. These systems have been verified and deployed on Dawning 4000A super server for scientific computing and other cluster systems for business computing.

Besides, we have evaluated four main design methodologies in term of the proposed four evaluation criteria. Our practices and qualitative analysis show our design methodology of Phoenix system has advantages over others in terms of proposed criteria.

In the near future, we will propose complete performance evaluation criterion for management service framework and evaluate performance of Phoenix stack on the production platform.

8. ACKNOWLEDGMENTS

I wish to thank all the members of Parallel and Distributed System Group for implementing Phoenix cluster system software stack at the Institute of Computing Technology, The Chinese Academy of Science.

9. REFERENCES

[1] Stephen L. Scott，OSCAR and the Beowulf Arms Race for the “Cluster Standard,” Proceedings of IEEE Cluster 2001.

[2] Ewing Lusk, An Open Cluster System Software Stack,Recent Advances in Parallel Virtual Machine and Message Passing Interface, Volume 3241, 2004

[3] Component-Based Cluster Systems Software Architecture: A Case Study ,Desai, R. Bradshaw, A. Lusk, E. Lusk, and R. Butler, Proceeding of IEEE Cluster 2004

[4] Ralph Butler, Narayan Desai, Andrew Lusk, Ewing Lusk, The Process Management Component of a Scalable Systems Software Environment, Proceedings of IEEE Cluster 2003, Hong Kong.

[5] Rob Armstrong, Dennis Gannon etc, Toward a Common Component Architecture for High-Performance Scientific Computing, , Proceedings of HPDC 1999

[6] Dennis Gannon, Sriram Krishnan etc, On Building Parallel & Grid Applications: Component Technology and Distributed Services , Cluster Computing, Volume 8, Number 4 ,2005

[7] http://www.openpbs.org

[8] Sameh A. Fakhouri, Germ´an Goldszmidt, Michael Kalantar, John A. Pershing, GulfStream – a System for Dynamic Topology Management in Multi-domain Server Farms, Proceedings of IEEE Cluster 2001

[9] Fay Chang etc , Bigtable: A Distributed Storage System for Structured Data, Fay Chang, OSDI 2006

[10] Mike Burrows etc, The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

[11] Jeffrey Dean etc , MapReduce: Simplified Data Processing on Large Clusters, Sanjay Ghemawat, OSDI 2004

[12] Helen J. Wang, John C. Platt, Yu Chen, Automatic Misconfiguration Troubleshooting with PeerPressure, OSDI 04

[13] Federico D. Sacerdoti, Mason J. Katz, Matthew L. Massie, David E. Culler, Wide Area Cluster Monitoring with Ganglia, In Proceedings of the IEEE Cluster 2003 Conference, Hong Kong.

[14] Ni Guangbao, Ma Jie, Li Bo, GridView: A Dynamic and Visual Grid Monitoring System, HPCAsia 04

[15] E. Moreira, G. Almási, C. Archer, R. Bellofatto, P. Bergner etc, Blue Gene/L programming and operating environment, IBM Journal of Research and Development, Vol.49, No.2/3, 2005

[16] Y. Aridor, T. Domany, O. Goldshmidt, J. E. Moreira, and E. Shmueli, Resource allocation and utilization in the Blue Gene/L supercomputer, Vol.49, No.2/3, 2005

[17] Jean-Charles Tournier, Patrick G. Bridges etc, Towards a Framework for Dedicated Operating Systems Development in High-End Computing Systems, ACM SIGOPS Operating Systems Review, Volume 40 , Issue 2, April 2006

[18] P. Uthayopas, S. Phatanapherom, T. AnGSDkun, S. Sriprayoonsakul, "SCE: A Fully Integrated Software Tool for Beowulf Cluster System," in *Proceedings of Linux Clusters: the HPC Revolution*, National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana, Illinois, June 25-27, 2001.

[19] Atsushi, SCore: An Integrated Cluster System Software Package for High Performance Cluster Computing, Proceedings of IEEE Cluster 2000.

[20] Clemens Szyperski. Component Software: Beyond Object-Oriented Programming, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., 2002.

[21] Mary SHAW, DAVID GARLAN, software architecture: perspective on an emerging discipline, prentice hall 1996.

[22] Zhihong Zhang, Dan Meng, Jianfeng Zhan etc, Easy and Reliable Cluster Management: The Self- management Experience of Fire Phoenix, IPDPS 2006 SMTPS workshop

[23] P. Papadopoulos, M. Katz, and G. Bruno, “NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters,” Proceedings of IEEE Cluster 2001

[24] Jianfeng Zhan, Ninghui Sun, Fire Phoenix Cluster Operating System Kernel and its Evaluation, Proceeding of IEEE Cluster 2005, Boston, MA, USA.

[25] Bibo Tu, Ming Zou, Jianfeng Zhan etc, Design Patterns of Scalable Cluster System Software, IEEE PDCAT 2006.

[26] Dawning 4000A, http://top500.org/system/7036, 2004

[27] Jianfeng Zhan, Lei Wang etc, A Layered Design Methodology of Cluster system software stack, Poster, International Conference on Cluster Computing, Austin, 2007