Front.Comput.Sci.
DOI

REVIEW ARTICLE

Consolidated Cluster Systems for Data Centers in the Cloud Age: a Survey and Analysis

Jian LIN (✉)[1,2], Li ZHA [1], Zhiwei XU [1]

1 Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
2 Graduate University of Chinese Academy of Sciences, Beijing 100049, China

Abstract In the cloud age, heterogeneous application modes on large-scale infrastructures bring about the challenges on resource utilization and manageability to data centers. Many resource and runtime management systems are developed or evolved to address these challenges and relevant problems from different perspectives. This paper tries to identify the main motivations, key concerns, common features, and representative solutions of such systems through a survey and analysis. A typical kind of these systems is generalized as the *consolidated cluster system*, whose design goal is identified as reducing the overall costs under the quality of service premise. A survey on this kind of systems is given, and the critical issues concerned by such systems are summarized as *resource consolidation* and *runtime coordination*. These two issues are analyzed and classified according to the design styles and external characteristics abstracted from the surveyed work. Five representative consolidated cluster systems from both academia and industry are illustrated and compared in detail based on the analysis and classifications. We hope this survey and analysis to be conducive to both design implementation and technology selection of this kind of systems, in response to the constantly emerging challenges on infrastructure and application management in data centers.

Keywords Data center, cloud computing, distributed resource management, consolidated cluster system, resource consolidation, runtime coordination

Received month dd.yyyy; accepted month dd.yyyy

E-mail: *linjian@ict.ac.cn*

1 Introduction

In the cloud age, both public data centers and enterprise private clusters have more or less encountered the following situations: coexistence of heterogeneous programming paradigms and multiple application frameworks; large-scale infrastructures and massive data or requests; variable application modes and workloads along with business requirements. Many challenges have arisen, such as resource utilization and manageability, which lead to the evolution of data center management mechanisms. A number of solutions [1–6], either complete systems or particular technologies, are proposed to solve the emerging challenges and relevant problems in this background. By observing some recent systems, we focus on a typical kind of systems characterized by global consolidated management. From the perspective of research, we try to identify the main motivations and common design ideas of these systems, and how to define, classify, and analyze such systems. From the practical point of view, we hope to promote the design and implementation of distributed resource management systems, and to help the data center administrators with technology selection.

Therefore, a survey on data center management systems with consolidation characteristics is presented in this paper. An analysis on such systems is performed to reveal the essential properties of these systems, and to identify the design goal, classification criteria, critical issues, and common features of them.

For practical purposes, an important criterion for the se-

lection of surveyed objects is that the system should be completely implemented and available for download or purchase. Pure research, prototype systems, and private solutions are not the main surveyed objects. However, forward-looking ideas and particular technologies proposed by such work are referred in the survey, in order to achieve a comprehensive coverage over relevant work, and be conducive to the advanced research and development of such kind of systems.

The rest of this paper is organized as follows. Section 2 presents the background of data center resource and runtime organization modes. Section 3 shows the motivation of researching and developing consolidated cluster systems. Section 4 proposes the critical issues concerned by such systems. Section 5 gives the analysis and comparison of five typical systems in detail. Section 6 lists the related work, and the last section concludes this paper.

2 Background

The approaches of infrastructure and application management of data centers often depend on the data center magnitude, business models, and sometimes historical reasons. Nevertheless, three typical resource and runtime organization modes are identified in current practices: *dedicated*, *hybrid*, and *consolidated*. Different organization modes differ in the relationship between infrastructures and applications, as well as the capability of resource and runtime control. For the dedicated organization, the application frameworks are built directly on their dedicated infrastructures, and the upper applications run independently in their own ways. For the hybrid organization, the infrastructures are shared and accessed directly by coexistent application frameworks with optional resource sharing and scheduling mechanisms. For the consolidated organization, both the infrastructures and applications are globally managed and coordinately scheduled by a suite of uniform resource and runtime control system.

Fig. 1 shows the differences among basic structures of the three resource and runtime organization modes, and illustrates the discrepancy on system consolidation properties [7]. Their concrete characteristics are detailed in the following subsections.

2.1 Dedicated Organization

Dedicated organization [8] is the most ordinary organization mode in a data center. Each application framework is built directly on its dedicated infrastructure, and independently occupies a set of physical resources. Without any additional mechanism, this organization manner is non-interfering with the applications, and naturally supplies performance isolation and fault isolation among different applications. On the other hand, dedicated resource may lead to low resource utilization and high power consumption [9]. Imbalanced application workloads will not be scheduled effectively in the global scope. As a result, it prevents service providers from benefiting from the economies of scale [10]. In this organization mode, the application owners should manage and maintain the applications and infrastructures in *ad hoc* ways by themselves, so it may cause some common management activities overlapped. Because this mode avoids a global hotspot, scalability is limited only by the application frameworks' capabilities. However, the high management costs and low flexibility determine that the dedicated organization mode is not suitable for the data centers of large-scale infrastructures or variable business.

2.2 Hybrid Organization

Hybrid organization [8, 11] means the infrastructures are shared and accessed directly by coexistent application instances of the same or different application modes. The most naive hybrid manner is to run different application frameworks on the same infrastructure, which improves resource utilization, but damages isolation and reliability. With the help of specific resource sharing and scheduling mechanisms or policies, the infrastructure can be controlled-shared among applications assuring isolation and reliability to some degree. Virtualization [12–14] is a common solution that makes a balance between resource sharing and isolation. It is suitable for scenarios without strict I/O performance demands or special hardware requirements.

Hybrid cloud [15] and other multi-source grid/cluster practices [16] can be regarded as another style of hybrid resource organization mode. They supply the upper applications with a variable resource capability transparently. From the cloud providers' view, their public resources are controlled-shared among different tenants. The coarse-grained isolation unit is often a virtual machine (VM) or a resource slot. From the application owners' view, the hybrid usage of private resources and public resources enables scalability and flexibility for their variable workloads. This business model benefits both the cloud providers' resource utilization and the application owners' resource costs.

In addition, there have been a number of distinctive researches in accord with the hybrid organization mode.

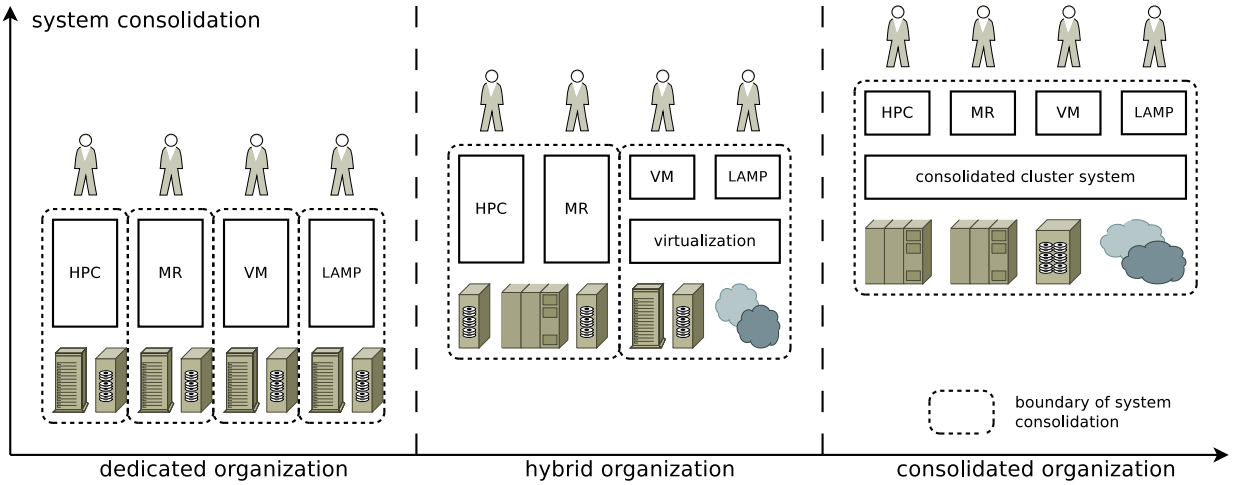

Fig. 1 Three typical resource and runtime organization modes in data centers. The abbreviations in the figure are explained as follows: HPC = high-performance computing, MR = MapReduce (a typical example of data computing), VM = virtual machine (a typical example of virtualized resource leasing), and LAMP = Linux+Apache+MySQL+Perl/PHP/Python (a typical example of traditional long-lived service).

Cluster-on-Demand (COD) [4] is a cluster management architecture to partition resources in mixed-use clusters, which supplies dynamic virtual clusters associating variable shares of resources to serve coexistent applications. VGrADS [5] proposes a virtual grid abstraction to unify the private cluster and public cloud, and supports e-science workflows with high resource utilization and fault-tolerance on distributed infrastructures. CometCloud [6] is an autonomic computing engine integrating dynamic grid and cloud infrastructures, which provides a common service layer and a programming layer for heterogeneous application management and development.

To sum up, hybrid organization focuses on efficient resource integration and sharing instead of detailed runtime control. The support for resource/workload heterogeneity and the capability of resource/workload control are often contradictory to a hybrid solution. Hybrid resource sharing and scheduling mechanisms and policies are generally non-interfering or little-interfering with the applications. It is friendly to the application owners, but may not achieve the best resource utilization and manageability.

2.3 Consolidated Organization

Consolidated organization means both the resource management and runtime control are uniformly orchestrated by a global system. In this paper, such a system is described as a *consolidated cluster system*. We define the consolidated cluster system as follows:

Definition 1. A consolidated cluster system is a type of platform-level software or middleware, which manages heterogeneous infrastructures and application workloads for a distributed computing environment in a logically uniform manner.

In this organization mode, diverse application modes and heterogeneous infrastructures are supported. The resource access of applications is not direct logically, but supervised by a consolidated cluster system. The system allocates resources to application frameworks according to its uniform policies. The runtime entities of application instances are controlled by the system to a certain extent. For gaining fine-grained runtime control, this organization manner is interfering with the applications in some cases.

The core design goal of a consolidated cluster system is to reduce the overall costs for operating a data center under the quality of service (QoS) premise, where the costs mainly derive from hardware resource occupation and manual application management. The concrete optimization objectives generally include resource utilization, manageability, scalability, reliability, flexibility, and so on. However, it is difficult to achieve all the objectives simultaneously. For example, global real-time resource usage information is necessary for obtaining the optimization objective of resource utilization, but a centralized information gathering and decision making mechanism will limit the system scalability. Therefore, the design of a consolidated cluster system often involves trade-offs, and a consolidated cluster system usually has its well defined target application scenarios.

To cope with the requirements of data center management in the cloud age, especially to response to the challenges of “big data” [17] in the Internet-scale computing scenario,

several consolidated cluster systems are proposed, such as Apache Mesos [1] and the next generation of Apache Hadoop [2]. Some traditional cluster computing technologies derived from grid and high-performance/high-throughput computing, like Condor [18] and Oracle Grid Engine [19], are evolving to adapt to the burgeoning application scenarios and becoming consistent with the definition of consolidated cluster systems. As will be referred in Section 6, researches on consolidated solutions are also springing up.

3 Motivation

The emergence of consolidated organization mode reflects the practical demands of data centers in the cloud age. The main motivations for design and development of consolidated cluster systems are summarized and discussed in this section.

3.1 Heterogeneous Application Modes

Along with the widespread application of computing technology in all walks of life, a variety of business requirements have emerged. To address the challenges from different application scenarios, diverse application modes and corresponding programming models [20] are continually proposed and implemented, or inherited from the traditional computing environment. For large-scale computing infrastructures of either a public data center or an enterprise private cluster, coexistence of heterogeneous application frameworks is a common situation. The issues of concern include scalability of a single framework, isolation of coexistent frameworks, and flexibility for introducing new frameworks.

Application modes in data centers can be classified according to different criteria. From the perspective of runtime span, they can be divided into long-lived services and one-time tasks. From the perspective of responsiveness, they can be divided into online/real-time applications and offline/batch processing applications. Furthermore, application modes can be classified by resource preferences, runtime preemption, and other taxonomic characters. Differentiated application requirements and the conflicts among them are inevitable. These complex factors make it a challenging job to coordinate heterogeneous application modes in a single administrative domain.

This paper focuses on the heterogeneous application scenarios involving the following application modes, which are considered to be widely adopted in the data centers of the cloud age.

Data computing (DC). Big data has become a key basis of innovation, competition, and productivity in many fields of the global economy nowadays [17]. Several data computing paradigms and frameworks are proposed. For example, MapReduce [21] and its open-source implementation in Apache Hadoop [22] for batch processing, and Google Percolator [23] and Yahoo! S4 [24] for stream computing.

High-performance computing (HPC) and high-throughput computing (HTC). Traditional HPC/HTC application modes will remain in data centers for scientific computing and other high-end applications or legacy systems. One of the classic programming paradigms of high-end computing is Message Passing Interface (MPI) [25]. Lots of implementations are available, such as MPICH2 [26] and Open MPI [27].

Virtualized resource leasing. Resource provisioning on demand is a key feature of cloud computing, and resource leasing via virtualization is the main business model in public clouds [20,28]. Note that virtualization here does not only represent virtual machines, but also cover all the leasable virtual private resources in spite of their existing forms. Traditional business models like web hosting and dedicated server hosting will benefit from the resource leasing model because of its profits on resource utilization and manageability.

3.2 Resource Utilization

The main resource types in data centers include computing resource, storage resource, and network resource. Different applications have different patterns of resource usage. In a heterogeneous application scenario, resource usage problem presents for several reasons. From the perspective of resource provisioning, the main cause is uncertain resource planning. Predetermined resource planning may not suffice the resource requirements in practice. Fixed resource allocation may lead to resource waste or shortage. From the perspective of application execution, the main cause is unbalanced resource access. Without appropriate resource placement or runtime scheduling, different applications may access one type of resource intensively at the same time, while keeping another type of resource idle.

Each type of resource takes on its specific utilization issue. For the computing resource, appropriate application deployment and runtime scheduling will utilize the locality feature of processor and memory to improve the overall performance [29]. For the storage resource, data locality needs to be optimized to minimize data migration and remote access [30]. Data compression and reuse are also beneficial in improving

storage utilization. For the network resource, link utilization is often the key optimization point [31], and the ratio of effective payload determines the bandwidth utilization.

The heterogeneity of infrastructure is another important challenge for resource utilization. For one thing, servers and network devices in one cluster can have heterogeneous capabilities since making use of previous investment [32]. A distributed application framework with the assumption of resource homogeneity can cause unbalanced resource utilization and therefore degraded performance [33]. For another, special or customized hardware, such as general-purpose graphics processing units (GPGPU) [34], and high-speed network adapters [35], are adopted and applied in some actual environment. It is requisite for a resource management system to be aware of the hardware specificity and map particular applications to the corresponding hardware, so as to maximize the utilization of specific resources.

Power consumption is a correlative problem raised by resource utilization. It is not only determined by the hardware efficiency, but also dependent on the capability of resource management systems and the efficiency of applications [9]. Energy-efficient management of resources is a hot research topic in the background of energy crisis. Furthermore, low resource utilization has been considered as the origin of some other cost problems in data centers [36].

3.3 Manageability

Manageability is a comprehensive problem [37–39]. Only the issues in reference to the consolidated organization mode are discussed in this subsection.

Single system image (SSI). Different application frameworks possess their own resource space, user space, and information space. For a large-scale data center, a single system image unifies the system view presented to applications, and then benefits several scenarios in development and deployment of heterogeneous applications. For example, a single user space enables the uniform resource leasing mechanism, and a single data view contributes to the data sharing and reuse among collaborative frameworks.

Common services. Underlying common services, such as system monitoring, resource accounting, and access control, are essential to the infrastructure and application management of data centers. Although many application frameworks have integrated their respective peripheral services, it is an ideal form to employ a suite of uniform common services in a single administrative domain. On the one hand, it reduces the construction and maintenance costs of redundant systems, and on the other hand, it enables realizing global management and optimization policies.

Interactive complexity. From the administrators' point of view, the manageability problem manifests in the number of managerial operations and corresponding responsibilities. In a naive hybrid cluster with m application frameworks and n resource sets, it needs $O(mn)$ operations to configure all the frameworks without a uniform mechanism. The responsibilities of infrastructure management and application management are interweaved [Fig. 2 (a)]. However, leveraging a uniform consolidation mechanism, it needs only $O(m + n)$ operations to configure all the frameworks, and the responsibilities of infrastructure management and application management are separable [Fig. 2 (b)].

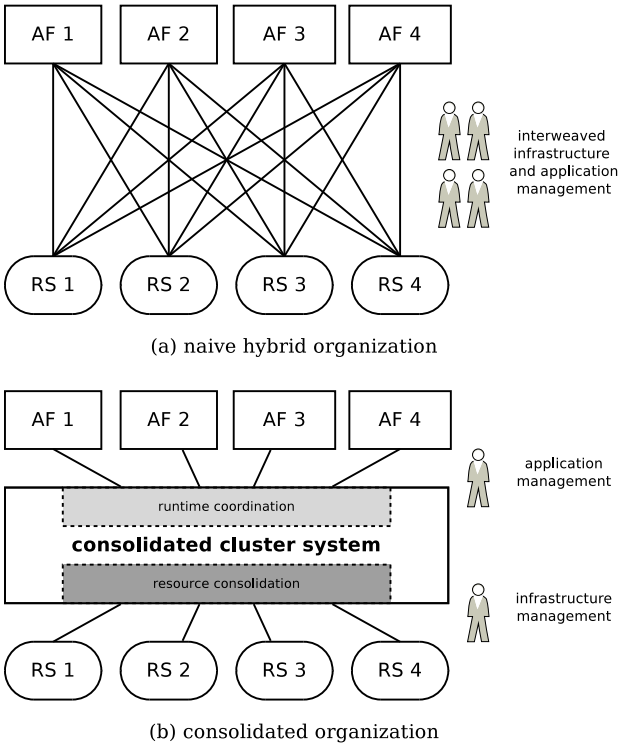

Fig. 2 The comparison of naive hybrid organization and consolidated organization on interactive complexity. The abbreviations in the figure are explained as follows: AF = application framework, and RS = resource set.

4 Critical Issues of Consolidated Cluster Systems

A consolidated cluster system works as the core system software of a data center. Like an operating system of a data center, it manages the underlying infrastructures and supports the upper applications from a global perspective. As shown in Fig. 2 (b), we consider the interactions among a system and

its managed infrastructures and applications. The functionality and responsibility of a system are divided into underlying *resource consolidation* and upper *runtime coordination*. In this section, we focus on these two elements. The issues and concerns of them are illuminated. A set of classification criteria are specified according to the main design styles and key external characteristics. A series of common features that should be attended when designing and implementing a consolidated cluster system are discussed as well.

4.1 Resource Consolidation

In the context of consolidated cluster system, resource consolidation is defined as follows:

Definition 2. Resource consolidation represents the relationship and corresponding mechanisms between a consolidated cluster system and the underlying infrastructures. It provides the approaches of global resource organization and management.

To achieve the system design goal, resource consolidation aims at controlling the resource occupation costs. In a heterogeneous application environment, its main work is to allocate a certain amount of resources to different application frameworks at an appropriate time according to specific rules [40, 41]. The concrete tasks often include resource planning, provisioning, scheduling, and monitoring. The optimization objectives and evaluation indicators of resource consolidation usually include resource utilization, resource provisioning overhead, infrastructure scalability, and responsiveness to variation of resource requirements [42].

From the perspective of resource scheduling opportunity, the resource consolidation approaches can be classified as follows:

- **Static**: resources are allocated at once or reserved in advance by a fixed amount before an application launching. Once the application instance has been executed, its resources cannot be scaled beyond the reservation.
- **Semi-dynamic**: resources are allocated or reserved by a certain amount before an application launching. When triggered by some predefined policies or manual operations, the resources can be scaled on demand.
- **Dynamic**: resources are not allocated or reserved when an application launches, but dynamically allocated and revoked during the runtime according to specific resource scheduling policies.

From the perspective of resource-workload relationship, the manners of resource consolidation can be classified as follows [43]:

- **Resource allocation**: the right of resource access is assigned to the requester, and the requester gains the control over resource for a limited time. The logical entity movement direction is "resource → requester". A typical example is the virtual machine provisioning in cloud computing.
- **Workload delegation**: the responsibility of workload execution is delegated to the executor, namely the resource provider, and the use of resource is controlled by the executor. The logical entity movement direction is "workload → executor". A typical example is the process of job scheduling in batch process systems.

From the perspective of resource isolation level, the isolation unit of resource consolidation can be classified as follows:

- **Process**: local OS processes provide the simplest way for resource isolation, and meanwhile the local OS account mechanism provides security isolation to a certain extent. Resource limitation should be enforced by kernel or external mechanisms.
- **OS isolation container**: OS isolation containers implemented by kernel mechanisms, such as Linux Containers [44], enable both resource isolation and limitation. These mechanisms combine the advantages of security and lightweight.
- **Virtual machine**: virtual machines can provide relatively rigorous resource isolation. Virtualization is widely adopted in cloud computing because of advantages like elastic deployment and online migration, though the overhead of virtualization is sometimes notable.
- **Physical machine**: physical machines provide a coarse-grained but rigorous way for resource isolation. For occasions of requiring special hardware or pursuing perfect performance, using physical machines is the best choice.

4.2 Runtime Coordination

In the context of consolidated cluster system, runtime coordination is defined as follows:

Definition 3. Runtime coordination represents the relationship and corresponding mechanisms between a consolidated cluster system and the upper application workloads. It provides the approaches of application runtime control and management.

To achieve the system design goal, runtime coordination aims at controlling the application management costs. In a heterogeneous application environment, its main work is to orchestrate all the runtime entities from different application frameworks to meet specific goals [1]. The concrete tasks often include launching, scheduling, monitoring, and revoking runtime entities like jobs and tasks. For a system with a coarse-grained runtime coordination manner, the elaborate internal runtime control can be executed by the application frameworks in their traditional ways. The optimization objectives and evaluation indicators of runtime coordination usually include overall performance, additional management overhead, runtime isolation, and fault-tolerance [42, 45].

From the perspective of policy decision point (PDP), the actions of runtime scheduling can be classified as follows:

- **Centralized**: the action of runtime scheduling is executed by a logically centralized point. Fine-grained schedulers within application frameworks may exist in this case, but they do not interact with the centralized scheduler. Replicas of the centralized scheduler for reliability or load balancing are allowed.
- **Hierarchical**: the action of runtime scheduling is collaborated by interactive entities of different execution levels, usually by a centralized scheduler from the consolidated cluster system with a set of distributed schedulers from application frameworks.
- **Decentralized**: the action of runtime scheduling is executed by decentralized entities, typically by the schedulers from application frameworks.

From the perspective of system-framework relationship, the runtime coordination approaches can be classified as follows:

- **Non-interference**: original application frameworks can be directly managed or integrated by the consolidated cluster system. The system encapsulates the frameworks optionally by external containers, and controls them via existent interfaces.
- **Interference**: application frameworks should be explicitly modified to collaborate with the consolidated cluster system. The frameworks need to integrate some mechanisms provided by the system in order to realize specific runtime coordination functionalities.

From the perspective of runtime scheduling granularity, the scheduling unit of runtime coordination can be classified as follows:

- **Application framework**: an entire application framework, including all the collaborative components and execution entities with necessary external containers. For example, a Hadoop instance.
- **Application instance**: an instance of a specific application workload, including all the distributed execution entities. For example, a Hadoop job.
- **Single executor**: an internal execution entity of an application instance, usually a local OS process. For example, a Hadoop task.

4.3 Common Features

The benefits derived from consolidated cluster systems manifest not only in their inherent functionalities, but also in the useful features provided to the existent application environment. These features supply the heterogeneous application frameworks with important supplements to the mismatched non-functional properties, and also supply the users with several essential values of the consolidated organization mode. Three main features that mainstream systems usually concern are identified and illustrated in this subsection.

4.3.1 Scalability

Scalability of a consolidated cluster system means its ability to accommodate the growth of managed infrastructures, workloads, and the magnitude of application frameworks. Comparing with the scale-up model in traditional high-performance computing, the scale-out model is emphasized in the cloud computing scenario, especially for data computing applications [46]. The concerns of scalability in such scenario depend on the division of responsibilities between the system and application frameworks. Generally, a consolidated cluster system designed for real-life business workloads should consider the increment of meta-information, control communication, and application failures. A simple metric for scalability is the maximum magnitude of managed nodes and data.

The architecture design will affect the scalability of a consolidated cluster system. For example, to achieve the targets of global management and scheduling optimization, some centralized components are indispensable. The complexity and density of concentrated interactions limit the potential of scale-out. Decentralized architecture with lightweight and distributed interactions is advantageous to scale-out, which makes a trade-off between the scalability of system and the optimal solutions of global resource consolidation and runtime coordination.

4.3.2 Reliability

Reliability of a consolidated cluster system means the ability to perform the required functionalities under certain conditions for a specified period of time. Specifically, two key features in reference to reliability, fault-tolerance and isolation, are concerned in this paper.

The fault-tolerance feature roots in the basic fact: the hardware and software failures are very common in a large-scale cluster [21]. Both the consolidated cluster system and the upper application frameworks may implement fault-tolerance. A well designed system should not introduce a new vulnerable link to the existent application environment. Many new programming paradigms and corresponding application frameworks designed for large-scale Beowulf clusters have taken account of fault-tolerance. However, some traditional parallel processing frameworks assume the reliability of infrastructure. In this case, a consolidated cluster system can provide external mechanisms, such as process checkpoint and restart, to enhance the reliability of particular application modes.

The isolation feature concerns fault isolation and security isolation among the system and all the managed application frameworks. In a production environment, it is necessary to guarantee a runtime failure or a security threat from one application framework not to affect the other application frameworks or the underlying consolidated cluster system. The determinants of isolation include the system-framework relationship and the isolation unit of resource consolidation. In order to balance reliability and resource utilization, trade-offs between resource isolation and sharing are often considered.

4.3.3 Flexibility

Flexibility of a consolidated cluster system is incarnated in the following aspects.

First, for the design of a general-purpose system, a challenge is to response the continual emergence and evolution of application modes and corresponding programming paradigms. Flexibility is represented in the ability of supporting new application frameworks [47]. To bring in a new application mode, a flexible consolidated cluster system requires minimal modification to both the system and the application framework. Furthermore, a flexible system may provide standardized mechanisms for application framework integration.

Second, for a production data center environment, the versatility of business and workloads often demands reconfiguration or redeployment of application frameworks. Flexibility is represented in the ability of responding the variation of business or workloads [48]. For the application frameworks with elasticity or scale-out features (*e.g.*, Hadoop), a well designed consolidated cluster system should supply the underlying capability of resource alteration, which should not be an additional external resource restriction. This is the key factor for resource utilization and cost optimization, and the basic property for certain service-oriented application modes like virtual resource leasing.

Flexibility brings down the management costs by reducing the overhead of environment reconstruction. However, it is an apparent experience that flexibility is conflicted with the granularity and intensity of management. To implement a pragmatic consolidation system, trade-offs are unavoidable.

5 Examples of Consolidated Cluster System

Five typical consolidated cluster systems are surveyed in this section. The survey is comprehensive enough to cover both open-source and commercial systems, and from both academia and industry. Some systems are newly developed to response the challenges in the cloud age, while the others are evolved from solutions for the traditional application modes in data centers. The resource consolidation and runtime coordination technologies of different systems are extracted and analyzed. A summary and comparison of the typical systems is presented at last.

5.1 Apache Mesos

Apache Mesos [1] is an open-source cluster manager providing efficient resource isolation and sharing across distributed application frameworks. It originated from a research project at University of California, Berkeley, and then joined the Apache Incubator [49] in 2011. Mesos is inspired by the springing up of diverse cluster programming frameworks. The target of this project is to effectively organize the runtime of different application frameworks and other long-lived services on shared infrastructure. Mesos also aims at facilitating the development of new distributed programming frameworks.

The Mesos team compared two common solutions for supporting heterogeneous application modes on shared cluster: static resource allocation by physical machines, and dynamically shared infrastructure by fine-grained resource scheduling. They believed that the first solution is easy to be integrated with existent application frameworks, but the resource

allocation granularity is too coarse to fit the runtime scheduling unit (*e.g.*, a slot) of the application frameworks. Mesos chooses the second solution that is to execute tasks from different application frameworks on the shared infrastructure with a uniform resource scheduling mechanism. The main challenge for this solution is that existent application frameworks are not designed to perform with the fine-grained resource across framework, so modifications to their original implementation are necessary.

The architecture of Mesos is shown in Fig. 3. A Mesos *master* node manages a group of Mesos *slave* nodes on which the tasks of applications run. Each application framework consists of a *scheduler* registered with the master, and a set of *executor*s running on the slaves. The design philosophy of Mesos is to define a minimal interface that enables efficient resource sharing, and delegate the control over task scheduling to application frameworks. This manner allows the frameworks to deal with their specific problems subtly, and keeps Mesos simple in order to realize scalability and reliability.

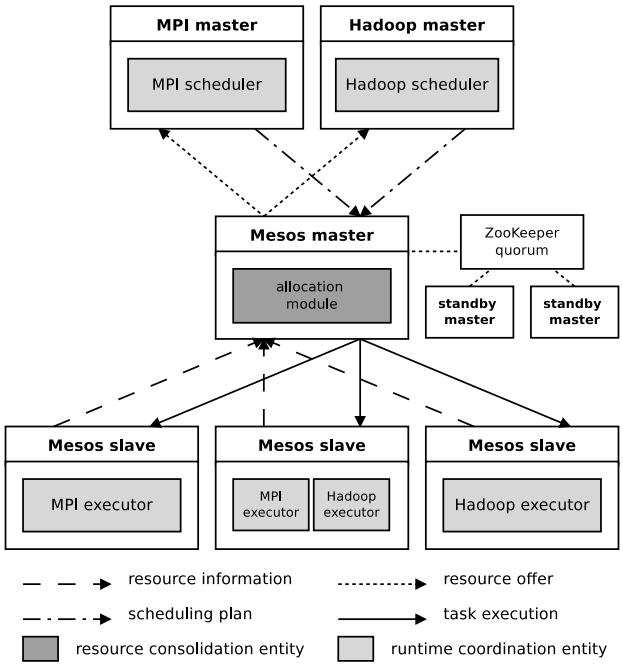

Fig. 3 The architecture and resource offer workflow of Mesos

Resource consolidation and runtime coordination are implemented by a distributed two-level scheduling mechanism called *resource offer*. The master executes the lower-level resource scheduling, which realizes resource consolidation. It receives available resource information from the slaves, and sends resource offers to the modified application frameworks. The application frameworks execute the upper-level task scheduling, which realizes runtime coordination. They choose resources from the resource offers, and reply the master with their resource requirements and task scheduling plans. Then, the master sends the tasks of application instances to accordant executors. The resource offer process is a dynamically repeated process accompanied by the start and finish of tasks. The scheduling mechanism of Mesos particularly considers the heterogeneous capabilities of resources, so that it can provide fairness to the application frameworks with the homogeneity assumption [33, 50]. To sum up, the isolation unit of resource consolidation is the local process of a task, and the scheduling unit of runtime coordination is a task of an application instance. Though Mesos implements a hierarchical decision model, the scheduling performance, including framework ramp-up time, job completion time, and system utilization, is good in practice.

Since the fine-grained scheduling is application related, the two-level mechanism must solve some subtle problems in reference to specific application frameworks. Rejection and filter mechanisms are introduced to optimize the performance of resource offer matching for application frameworks with complex resource constrains. Delay scheduling [30] is employed to avoid the confliction between fairness and data locality for data-intensive application frameworks like Hadoop. Revocation mechanism is used for resource reallocation in the priority scheduling policy, which distinguishes between fault-tolerant and fault-sensitive application modes.

Mesos is a flexible platform that provides standardized APIs for programming paradigm integration. Mainstream application frameworks including Hadoop and MPICH2 are imported with necessary modifications. A new application framework optimized for iterative computing called Spark [51] is developed by the Mesos team to prove the advanced development potential of Mesos. Indiscriminate slaves enables flexible multiplexing of heterogeneous application workloads, and scalability is ensured by the distributed resource offer mechanism, which has been verified by both emulation and production application. For reliability, the Mesos master is replicated by ZooKeeper [52], and the state of master is designed to be a reconstructible soft state. As an optional feature, Mesos provides security and fault isolation among framework executors on the same slave by using OS isolation containers like Linux Containers.

5.2 Next Generation of Apache Hadoop

Apache Hadoop [22] is an open-source MapReduce-based distributed computing framework with a distributed file sys-

tem maintained by the Apache Software Foundation, and supported by Yahoo! and other companies. It has achieved great success in the field of massive data processing. To solve the problems found in practice, and to address the challenges from newly emerging application requirements, the next generation of Apache Hadoop framework [2], also known as Yet Another Resource Negotiator (YARN) or MapReduce 2.0 (MRv2), has been proposed in early 2011, and integrated into the Hadoop 2.0 (previously known as Hadoop 0.23) series in 2012.

In the original Hadoop architecture, the JobTracker is a central service in charge of both resource management and job scheduling for a cluster. With the evolution of large-scale data computing applications, several technical deficiencies have been identified, such as large memory consumption, unreasonable threading model, and the limitations of scalability and reliability [53]. The primary idea of YARN is to split up the two core functionalities from the original JobTracker. This means to decouple runtime coordination from resource consolidation. Moreover, an important target of YARN is to support distributed programming paradigms other than MapReduce, and to support limited, short-lived services. These indicate that Hadoop is evolving into a consolidated cluster system.

As shown in Fig. 4, the architecture of YARN consists of a global *ResourceManager* and a set of per-node *NodeManager*s. The ResourceManager is the ultimate resource scheduler in the system, which organizes the global assignment of computing resources to application instances. It is composed of a *Scheduler* and an *ApplicationsManager*. The ApplicationsManager accepts job submissions and negotiates the first set of resources for job execution, and the Scheduler delegates the application instances to the execution nodes. The NodeManagers run as the local agents on the execution nodes, which set up the runtime environment including tasks' binaries and libraries, control and monitor the distributed resources and workloads, and communicate with the global Scheduler.

Resource consolidation and runtime consolidation are realized by a two-level architecture. At the lower resource level, the use of resources is controlled by the *resource container* mechanism. A resource container is a runtime abstraction that incorporates the resource constraints such as processors, memory, storage, and so on (in the initial versions, only memory is considered). The resource constraints are enforced by a process monitoring mechanism at the NodeManager. All the resource containers are indiscriminate so that every chunk of equivalent resource is fungible. Comparing with the simple slot abstraction in the original Hadoop architecture, it can alleviate the performance degradation derived from the homogeneity assumption by fine-grained scheduling. At the upper runtime level, a per-application *ApplicationMaster*, which is a framework-specific library, manages an application instance's life cycle. The ApplicationMaster is responsible for negotiating resource containers with the Scheduler, launching and managing the task executions according to specific application modes, and monitoring or tracking the runtime status of an application instance. In this way, the complexity of heterogeneous programming paradigms is separated from the system core.

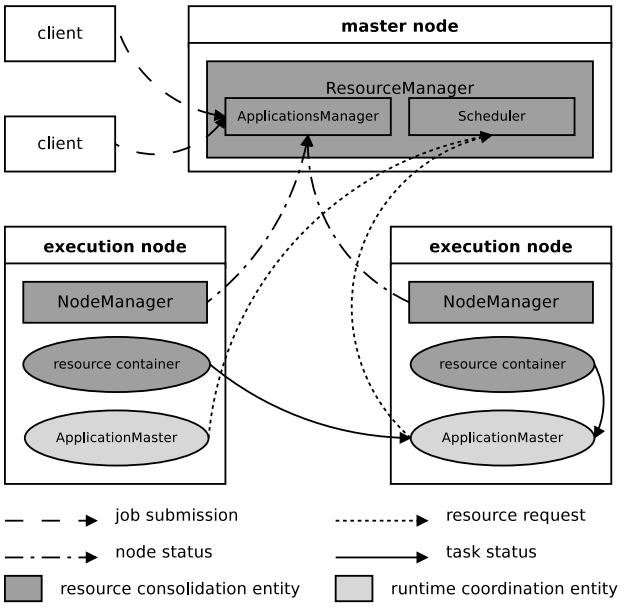

Fig. 4 The architecture and main workflow of YARN

YARN plans to support various application frameworks. MapReduce is naturally supported. Other frameworks being imported include Spark for iterative computing, Apache HBase [54] for large-scale semi-structured data processing, Apache HAMA [55] for bulk synchronous parallel (BSP) model, Apache Giraph [56] for graph processing, and Open MPI for high-performance computing. Standardized interfaces for framework integration are provided. To collaborate with the uniform two-level scheduling architecture, modifications to the existent frameworks are necessary. Pluggable component mechanism is widely accepted for flexibility, such as Scheduler plug-ins [2] allowing different resource partitioning policies for various application modes, and serialization plug-ins [57] enabling wire compatibility.

Scalability is a design goal of YARN. The target of the initial versions is to support a cluster of 6,000 nodes and 100,000 cores. Comparing with the original Hadoop model,

the YARN model extremely tightens the amount of state and passed information. For reliability, ResourceManager stores its state in ZooKeeper, so that the global information can be quickly recovered after a failure. The failure of an application instance can be recovered as well from saved persistent state by the corresponding ApplicationMaster. As an evolutionary system for industrial applications, YARN also concerns many practical problems, such as the compatibility with different versions of Hadoop, and the ability to control custom upgrades to software stacks.

5.3 Condor

Condor [18] is a classic open-source high-throughput computing software framework. It has been developed by University of Wisconsin-Madison since 1980's. The Condor project originally focuses on customers with large computing needs and environments with heterogeneous distributed resources. It provides the mechanisms of job queueing, job scheduling, resource monitoring, resource management, and so on. In recent years, the Condor community pays attention to uniform management and interoperation with some new-style distributed computing frameworks, and achieves good practical effects. So Condor can be considered as a consolidated cluster system now.

The composition of Condor is shown in Fig. 5. A cluster managed by Condor is called a Condor *pool*, where a single machine runs as the *central manager*, and the other machines run as *regular node*s (may be *submit-only node*s or *execute-only node*s). A group of daemons run on different machines. The *master* daemon spawns and monitors the other daemons, including the *collector* daemon collecting information from the other daemons, the *negotiator* daemon performing matchmaking between the resources and jobs, the *schedd* daemon maintaining job queues, and the *startd* daemon managing job executors. Inheriting the traditional batch processing system, Condor defines a *job* as the runtime scheduling unit, and simply abstracts a computer for executing jobs as a *machine*. For a classic batch processing application, the collector periodically obtains the queue information from submit nodes and the resource information from execute nodes. When a job is submitted to a schedd, the request is transferred to the negotiator by the collector. The negotiator matches the request with available resources, and then asks for an appropriate startd to execute the job.

Condor raises the conception of *universe* to manage different execution environments for heterogeneous application modes. Predefined universes include vanilla, standard, parallel, grid, VM, and so on. They define different functional constraints. The vanilla universe is a general environment for any batch-ready applications. It provides a few batch processing services like automatic file transfer. The standard universe provides a full-featured batch processing environment, which aims to support "the standard Condor application mode". Existing batch processing applications can be imported to this universe by relinking with Condor's libraries. Many advanced services, including transparent job checkpoint, migration, and remote I/O mechanism, are available so that uniform runtime management policies can be actualized. To support a variety of parallel computing frameworks, the parallel universe is designed. It coordinates the execution of jobs which need to be co-scheduled. MPI and MapReduce have been imported to this universe [58, 59]. With the help of grid agent components (*e.g.*, Condor-G/C), the grid universe offers a choice for interoperating with diverse grid computing frameworks, including Globus [60], Portable Batch System (PBS) [61], and even another Condor pool. It provides a scalable hierarchical extension that allows Condor running as a central scheduler, and heterogeneous grid middleware as executors. To support virtualized resource leasing, The VM universe is introduced. It launches a virtual machine instance as a job, providing virtual machine checkpoint, migration and other functionalities. Hypervisors including Xen [12], VMware [13], and Kernel-based Virtual Machine (KVM) [14] are supported.

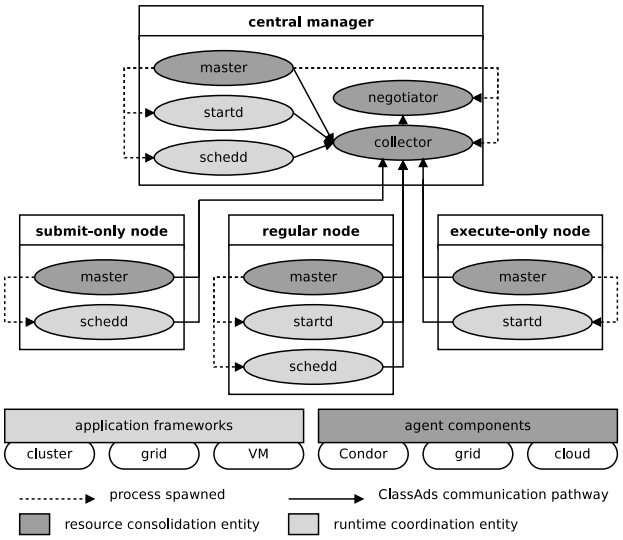

Fig. 5 The composition of a typical Condor pool

For resource consolidation, the isolation unit is the local process of a job in most universes. Condor introduces the *ClassAds* [62] mechanism for describing both the jobs' re-

source demand and the machines' resource supply, which involves hardware capabilities, specific device configurations, geographical location, *etc*. Matchmaking based on ClassAds is the core work of Condor, which delegates workloads to the best fitted executors. This mechanism allows both parties to implement sophisticated resource scheduling algorithms. Static resource constraint is implemented by enforcing the declarations in ClassAds, but not by real-time monitoring.

Runtime coordination of Condor is coarse-grained and non-interfering. For application frameworks in the parallel universe, it provides wrappers or assistant programs to control the execution of jobs. Condor schedules the jobs centrally, and the application frameworks manage the runtime behaviors of jobs (such as task scheduling) in their conventional manners. Security isolation in execute nodes is enforced by the account and chroot mechanisms of local OS. Condor also provides a programmable workflow control mechanism called *DAGMan* [63]. It runs as a meta-scheduler to coordinate the execution of heterogeneous job sets with directed acyclic graph (DAG) dependencies.

The style of Condor's resource consolidation and runtime coordination originates its motivations: high-throughput computing and opportunistic computing. High-throughput computing emphasizes providing large amounts of fault-tolerant computational capacity over a long period of time by effectively utilizing all the available resources [64]. Thus, Condor is designed for running long jobs and scheduling resources at a macro scale, while fine-grained control contributes little to the goal. Opportunistic computing is to utilize the preexisting non-dedicated resources under distributed ownership whenever they are available. The architecture, location, and availability of resources are various, so least assumptions should be supposed, and least constraints should be imposed. These motivations also lead to the philosophy of Condor: flexibility [47]. New application frameworks are supported by configuring existent universes, and ClassAds are extensible for new frameworks. Because of these benefits, Condor has exceeded its original planned application modes today. New usages like running Condor as a cloud engine [43,65] are springing up. The advantage of the Condor's philosophy makes it an energetic system for decades.

5.4 Oracle Grid Engine

Oracle Grid Engine (OGE) [19], previously known as Sun Grid Engine (SGE), is a distributed resource management system developed by Sun Microsystems and acquired by Oracle Corporation. It is a widely used solution in the industry, especially in the field of high-performance computing. The original open-source Sun Grid Engine has become a commercial project since being acquired. Free and open-source successors of SGE, such as Open Grid Scheduler [66] and Son of Grid Engine [67], are available. However, some features discussed in this subsection are only available in the Oracle's commercial release.

The main application mode that OGE focuses on is batch processing. It supports jobs of many styles, including interactive jobs, parallel jobs, array jobs, and so on. OGE is responsible for managing the remote and distributed execution of user jobs, and scheduling the distributed resources such as processors, storages, and software licenses. OGE also provides a flexible plug-in mechanism for integrating a variety of application frameworks. It can manage long-lived service clusters as a part of load balancer or key performance indicator (KPI) tuner. Data computing frameworks like Hadoop can be handled as well by the new version of OGE with the data locality optimization feature.

The architecture and main workflow of OGE [68] is shown in Fig. 6. An OGE cluster is composed of a *master machine* and a set of *execution machine*s. Optionally, a set of *shadow master machine*s are deployed as backups. A *qmaster daemon* runs on the master machine playing the role of core scheduler. *Execution daemon*s run on the execution machines for job launching and executing. *Shadow daemon*s run on the shadow master machines waiting for fault resolution. *Communication daemon*s run on all the machines for inter-component communication. The computing capacity of execution machines are divided into *slot*s. A slot is mapping to a CPU core for the normal configuration. *Job*s with their resource requirements are submitted to the qmaster daemon by client utilities (*e.g.*, *qsub*) or Distributed Resource Management Application API (DRMAA) [69] enabled applications. The job requests will be *queue*d in the qmaster, and scheduled to appropriate execution machines in designated order according to specific *policies*. When a job is running, it can be suspended, resumed, or checkpointed owing to users' operations or system's policies. After a job has completed or failed, the execution daemon will notify the qmaster with the results.

As a grid middleware, OGE is a single access point to powerful distributed resources [70]. The isolation unit of resource consolidation is the local process of a job. The sharing of resources is restrained by the *ticket* mechanism, whereby jobs retrieve tickets from multiple policies to determine their relative priorities. As a complement to the initial static workload delegation, advance resource reservation is introduced

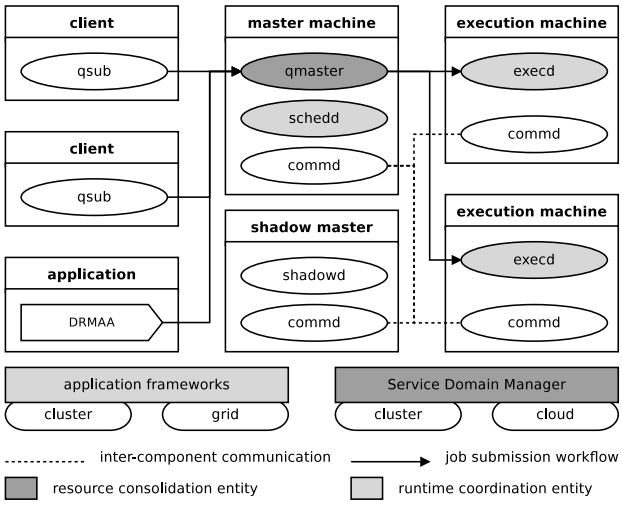

Fig. 6 The architecture and main workflow of OGE

to cope with the jobs with large resource needs. Designed for enterprise requirements, the resource accounting and access control mechanism of OGE is complete. On the one hand, it maps the roles of business management in the enterprise to the roles of resource control in the system, and on the other hand, it enforces the access control by using the security mechanism of local OS. Moreover, OGE introduces the conception of *complex* to provide exact information concerning the resource attributes, including global, host specific or queue related attributes. Complexes are taken into account during the processes of resource scheduling, bookkeeping, and capacity planning.

The batch queueing model is a classic runtime coordination model for batch job schedulers. In OGE, the scheduling unit of runtime coordination is a job. The applications are invoked via external interfaces without interference. For different application modes, users can choose the combinations of different queue types (batch, interactive, checkpointing, parallel, and transfer) flexibly. Delicate queue attributes, including execution methods, load thresholds, and subordinate queues, can be configured to fit for sophisticated application runtime requirements. Detailed scheduling policies can also be specified in OGE to satisfy the business needs of enterprises. Four types of high-level utilization policies are available: functional, share-based, deadline, and override.

OGE emphasizes scalability, which is designed and tuned for industrial-level and super-scale computing cluster [71]. An add-on component named *Service Domain Manager* [72] enables consolidating resources via service level objectives (SLO) from multiple clusters, either distributed physical clusters or virtual clusters provisioned by public clouds.

5.5 LingCloud

LingCloud [3] is an open-source cloud computing system software designed and implemented by Institute of Computing Technology, Chinese Academy of Sciences. The motivation of LingCloud is to solve the flexibility, manageability, and usability problems in the scenario of heterogeneous application modes coexistence on shared infrastructures. To effectively support this scenario, LingCloud follows the resource allocation manner in the "infrastructure as a service" (IaaS) model of cloud computing, and raises the conception of *Resource Single Leasing Point System* (RSLPS) for consolidating different infrastructures and applications by a uniform system. This conception has the following meanings: (1) single logic view which is independent with resource provisioning ways, (2) global management and operation through uniform interfaces, and (3) the operation scope involving both the infrastructures and the applications.

LingCloud focuses on five typical kinds of application modes, including large-scale data computing, high-performance computing, virtual resource leasing, data storage, and the raw application mode (which means the provisioned resources are maintained by applications directly). To consolidate multiple application modes, LingCloud proposes an *Asset-Leasing* model as shown in Fig. 7. It abstracts a resource user, either a person or a program, as a *tenant*, abstracts physical machines, virtual machines, and other leasable resources as *asset*s, and organizes the assets by *partition*s. The application frameworks are deployed on physical machines or virtual machines in certain partitions according to their characteristics. The limited ownership between a tenant and its assets is called a *lease*.

The isolation unit of resource consolidation is a set of assets, namely physical machines or virtual machines. An application framework exclusively occupies its leased assets in the corresponding partition. The main idea of partition is to organize a series of assets from the heterogeneous resource space according to hardware characteristics and application modes, and allow each partition adopting its unique management mechanism and policy. So the heterogeneity and specificity of hardware can be handled. Besides the first resource allocation by initial partitioning, assets' elastic flow among partitions is allowed as a semi-dynamic resource scheduling mechanism to support elastic computing frameworks. Flexibility is realized by creating a new partition type for a new application mode, and predefined general partition for the raw application mode is a universal container for simply integrating new application frameworks without specific functionali-

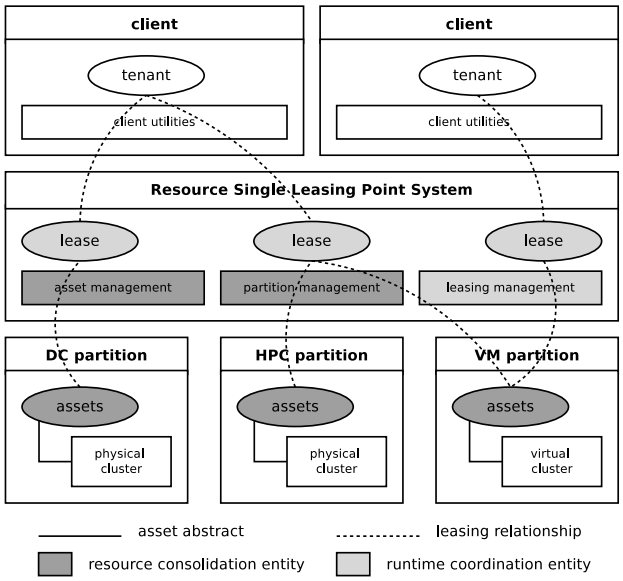

Fig. 7 The Asset-Leasing model of LingCloud

ties or policies.

In the LingCloud team's opinion, resource consolidation by the partition mechanism is an efficient way in the actual application. At first, hardware characteristics and application modes are tightly coupled in many cases. Deploying application frameworks on their accommodative hardware is a realistic choice in practice. Secondly, when the resources and workloads are consolidated according to their application modes, application-aware management policies are easy to perform, and some non-functional features, such as security and fault isolation, are easy to assure and measure. At last, partition being a strict isolation of application modes, it is friendly and flexible for application framework importation, and eliminates the interference on reliability among application frameworks. It makes LingCloud adoptable in the legacy application environment in operation.

The scheduling unit of runtime coordination is a lease. A lease represents a matchmaking process and a limited ownership, which provides appropriate assets with a preconfigured application framework to a corresponding tenant for running application instances. The application framework within a lease executes its runtime scheduling independently. From the macroscopic view, the runtime coordination decision is decentralized. Different leasing policies are provided to achieve different global optimization objectives. Predefined policies include high performance, low power consumption, custom location, and random. Advanced matchmaking policies can also be implemented by users. For implementation, LingCloud leverages Xen as the virtual machine hypervisor, and OpenNebula [73] as the virtual cluster controller. The toolkits from Open Source Cluster Application Resources (OSCAR) [74] are employed for physical machine management. Application encapsulation mechanism based on disk image is designed for fast deployment of partitions.

Since LingCloud is non-interfering with the applications, other external technologies are introduced to improve resource consolidation and runtime coordination. System monitoring mechanism based on ganglia [75] is used for collecting the resource usage information outside the application frameworks, in order to support resource related partition management policies. The grip (grid process) mechanism introduced from Grid Operating System (GOS) [76] is used for maintaining the leasing relationship, enabling advanced features for runtime control, such as single sign-on context [77] and security context [78].

5.6 Summary

Table 1 shows the comparison of the five typical systems on their basic properties, resource consolidation, runtime coordination, and other features. Mesos and YARN are the systems emerged in the context of big data in the cloud age. They follow the workload delegation model of batch processing, and schedule the heterogeneous runtime entities mixedly at the task granularity. The key idea is to consolidate workloads with different resource preferences on the same infrastructure so as to improve the overall resource utilization. The modification to existent application frameworks is a necessary price to pay. Condor and OGE are the systems evolved from traditional distributed resource management systems for the grid environment. They inherit the main architecture and manners of batch processing, and centrally schedule the runtime entities at the job granularity. They are usually non-interfering with the existent application frameworks which can be simply integrated and encapsulated. By contrast, LingCloud proposes the idea of Resource Single Leasing Point System based on the resource allocation model, which consolidates workloads with the same resource preferences on the same elastic partition, in order to improve the flexibility, manageability, and usability in heterogeneous resource/workload environments. Though different abstractions are presented, resource consolidation and runtime coordination are the critical issues concerned by the five systems. The relationship between the two sets of functionalities and responsibilities is reflected in the system architectures. All the systems consider the scalability, reliability, and flexibility features. Many

external mechanisms are provided to compensate for the mismatched non-functional properties of the heterogeneous application frameworks to a certain extent. The design goals of these consolidated cluster systems have been verified in practice.

6 Related Work

Some previous investigation and review work on grid and cloud computing have inspired us in this survey and analysis. A technical report from Berkeley [28] indicated the top obstacles and opportunities of cloud computing. We have analyzed these issues from the perspective of cloud computing providers, and believe that the consolidated cluster system is a feasible solution to the foundational software challenge. Foster *et al.* [20] compared the grid and cloud computing paradigms systematically. Based on the layered analysis, common problems including massive resource management and distributed computation coordination were identified. To study the critical issues and typical solutions of consolidated cluster systems, we refer to the problem definitions and related researches introduced in that paper. Krauter *et al.* [11] proposed a taxonomy and review of grid resource management systems (RMS). The authors presented an abstract structure of RMS. Based on the abstraction, classification criteria were organized in static (resources) and dynamic (scheduling) perspectives. The division of resource consolidation and runtime coordination is extended from this idea. Boutaba *et al.* [32] identified a key challenge in today's cloud environment, namely the heterogeneity of workloads and machine capabilities. Main causes and corresponding research questions were concluded by means of detailed analysis on production data centers. Our survey on systematic solutions and their survey on specific techniques are complementary.

Besides the typical consolidated cluster systems introduced in Section 5, there are many other related researches and systems with the idea of resource consolidation and runtime coordination. Steinder *et al.* [79] proposed a heterogeneous workloads automatic management architecture based on server virtualization technology. The core component of this architecture is an application placement controller, which coordinates the resource allocation and workload scheduling. Experiments in real environment and simulation involving a variety of batch job applications and web services demonstrate the effectiveness of automatic management mechanism in the heterogeneous application environment. Zhan *et al.* [41] proposed Phoenix Cloud, a system aiming at consolidating heterogeneous workloads on a uniform cloud computing platform. It provides a common service framework and cooperative resource provisioning and management policies. For different application modes, customized cloud management services should be developed as policy enforcers. Evaluations on heterogeneous scenario of web services and parallel batch jobs show that the system decreases the resource consumption, and meanwhile increases the number of completed jobs. Mateescu *et al.* [80] designed Elastic Cluster, a hybrid computing model combining the characteristics of both high-performance computing and cloud computing. It provides an elastic resource provisioning architecture consolidating computing capabilities from multiple private clusters and public clouds, and employs existent workload and resource management systems as execution controllers. Advanced resource management policies, including predictable execution and automatic scale-down, are implemented for optimizing scheduling performance and resource utilization. Wang *et al.* [10] concluded the resource provisioning and usage models in cloud computing, and proposed an innovative model, the Enhanced Scientific Public (ESP) model, for scientific communities to utilize elastic resources on a public data center while maintaining the users' flexible system control. By a model comparison and real-life experiments, they proved that the model with ideas of consolidation can make the users benefit from the economies of scale.

7 Conclusion

The challenges of heterogeneous application modes, resource utilization, and manageability along with the development of cloud computing applications have brought about the emergence of consolidated organization mode in data centers. We surveyed a series of systems developed or evolved under this background with the global consolidated management characteristic, and generalized them as consolidated cluster systems. The design goal of this kind of systems is identified as reducing the overall costs for operating a data center under the quality of service premise. The critical issues concerned by these systems are summarized as resource consolidation and runtime coordination. The two issues are analyzed and classified according to the design styles and external characteristics abstracted from the surveyed work. Three common features including scalability, reliability, and flexibility provided by mainstream consolidated cluster systems are observed. We analyzed and compared five typical systems from both academia and industry in the perspectives of resource

Table 1 Comparison of the example systems on their basic properties, resource consolidation, runtime coordination, and other features

Feature	Mesos	YARN	Condor	OGE	LingCloud
Origin	academia	community supported by industry	academia	industry	academia
License	Apache License 2.0	Apache License 2.0	Apache License 2.0	proprietary (previously SISSL[a])	Apache License 2.0
Supported application modes	HPC, offline DC, online DC, services, ...	HPC, offline DC, services, ...	HTC, HPC, offline DC, VM, ...	HPC, offline DC, services, ...	HPC, offline DC, online DC, VM, services, ...
Resource consolidation entity	system (resource offer)	system (ResourceManager)	system (ClassAds)	system (ticket)	system (RSLPS)
Resource scheduling opportunity	dynamic	dynamic	static	static	semi-dynamic
Resource-workload relationship	workload delegation	workload delegation	workload delegation (resource allocation for particular universes)	workload delegation	resource allocation
Isolation unit of resource consolidation	process (with optional OS isolation container)	process (with resource container)	process	process	physical machine or virtual machine
Runtime coordination entity	modified application frameworks	modified application frameworks	system and encapsulated application frameworks	system and encapsulated application frameworks	encapsulated application frameworks
Policy decision point	hierarchical	hierarchical	centralized	centralized	decentralized
System-framework relationship	interference	interference	non-interference	non-interference	non-interference
Scheduling unit of runtime coordination	single executor (task)	single executor (task)	application instance (job)	application instance (job)	application framework (lease)
Scalability features	distributed resource offer for elastically scaling	design and tuning for super-scale cluster	Condor-G/C for hierarchical extension	design and tuning for super-scale cluster	decentralized runtime coordination for elastically scaling
Reliability features	replicated masters by ZooKeeper, security and fault isolation based on OS containers	high-availability supported by ZooKeeper, security isolation based on OS mechanisms	job checkpoint and migration, security isolation based on OS mechanisms	shadow master machines, job checkpoint and restart, security isolation based on OS mechanisms	rigid security and fault isolation based on partitions
Flexibility features	standardized APIs for paradigm integration, indiscriminate slaves for flexible multiplexing	standardized APIs for paradigm integration, indiscriminate resource container for flexible multiplexing	universes for paradigm integration	plug-in mechanism for paradigm integration	partitions for paradigm integration, asset flow for elastic computing frameworks

[a] SISSL = Sun Industry Standards Source License

consolidation, runtime coordination, and the common features. The similarities and differences among systems of different backgrounds and motivations are concluded.

There is no perfect solution completely meeting the actual demands of a data center of particular operation mode. Integration and application of existent systems are often accompanied by customization and development of business-specific solutions. We hope this survey and analysis to be conducive to both design implementation and technology selection of consolidated cluster systems, in response to the constantly emerging challenges on infrastructure and application management from data centers in the cloud age.

Acknowledgements We would like to thank Xiaoyi Lu, Jie Zhang, Ruijian Wang, Xicheng Dong, Ying Wang, Chen Feng, Dixin Tang, Xiaona Liu from Institute of Computing Technology, Chinese Academy of Sciences, and Yongqiang Zou from Tencent Inc. for the valuable discussions and comments. We also thank the reviewers and editors for their feedbacks and suggestions. This research is supported in part by the National Basic Research (973) Program of China (Grant No. 2011CB302502, 2012CB316303), the Hi-Tech Research and Development (863) Program of China (Grant No. 2011AA01A203), and the Guangdong Talents Program (Grant No. 201001D0104726115).

References

1. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph A D, Katz R, Shenker S, and Stoica I. Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, 2011.
2. Murthy A C, Douglas C, Konar M, O'Malley O, Radia S, Agarwal S, and V V K. Architecture of next generation apache hadoop MapReduce framework. Technical report, Apache Hadoop community, 2011.
3. Lu X, Lin J, Zha L, and Xu Z. Vega LingCloud: a resource single leasing point system to support heterogeneous application modes on shared infrastructure. In: Proceedings of the 9th International Symposium on Parallel and Distributed Processing with Applications, ISPA '11, 2011, 99–106.
4. Chase J S, Irwin D E, Grit L E, Moore J D, and Sprenkle S E. Dynamic virtual clusters in a grid site manager. In: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, HPDC '03, 2003, 90–100.
5. Ramakrishnan L, Koelbel C, Kee Y, Wolski R, Nurmi D, Gannon D, Obertelli G, YarKhan A, Mandal A, Huang T M, Thyagaraja K, and Zagorodnov D. VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, 2009.
6. Kim H, el-Khamra Y, Jha S, and Parashar M. An autonomic approach to integrated HPC grid and cloud usage. In: Proceedings of the 5th IEEE International Conference on e-Science, e-Science '09, 2009, 366–373.
7. Lu X, Lin J, and Zha L. Architecture and key technologies of LingCloud. Journal of Computer Research and Development, 2011, 48(7):1111–1122.
8. Baker M, and Buyya R. Cluster computing at a glance. In: Buyya R, editor, High Performance Cluster Computing: Architectures and Systems, volume 2. Prentice Hall PTR, 1999, 3–47.
9. Beloglazov A, Buyya R, Lee Y C, and Zomaya A. A taxonomy and survey of Energy-Efficient data centers and cloud computing systems. In: Advances in Computers, volume 82. Elsevier B.V., 2011, 47–111.
10. Wang L, Zhan J, Shi W, and Liang Y. In cloud, can scientific communities benefit from the economies of scale? IEEE Transactions on Parallel and Distributed Systems, 2012, 23(2):296–303.
11. Krauter K, Buyya R, and Maheswaran M. A taxonomy and survey of grid resource management systems for distributed computing. Software: Practice and Experience, 2002, 32(2):135–164.
12. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, and Warfield A. Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP '03, 2003, 164–177.
13. VMware virtualization software. http://www.vmware.com/.
14. Kivity A, Kamay Y, Laor D, Lublin U, and Liguori A. KVM: the linux virtual machine monitor. In: Proceedings of the 9th Annual Ottawa Linux Symposium, OLS '07, 2007, 225–230.
15. Mell P, and Grance T. The NIST definition of cloud computing. Technical Report SP 800-145, Information Technology Laboratory, National Institute of Standards and Technology, 2011.
16. Silberstein M, Geiger D, Schuster A, and Livny M. Scheduling mixed workloads in multi-grids: The grid execution hierarchy. In: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, HPDC '06, 2006, 291–302.
17. Manyika J, Chui M, Brown B, Bugin J, Dobbs R, Roxburgh C, and Byers A H. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.
18. Litzkow M, Livny M, and Mutka M. Condor-a hunter of idle workstations. In: Proceedings of the 8th International Conference of Distributed Computing Systems, ICDCS '88, 1988, 104–111.
19. Oracle Corporation. Oracle grid engine: An overview. Technical report, 2010.
20. Foster I, Zhao Y, Raicu I, and Lu S. Cloud computing and grid computing 360-Degree compared. In: Proceedings of Grid Computing Environments Workshop, GCE '08, 2008.
21. Dean J, and Ghemawat S. MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th USENIX Symposium on Operating Systems Design & Implementation, OSDI '04, 2004.
22. Apache hadoop. http://hadoop.apache.org/.
23. Peng D, and Dabek F. Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the 9th USENIX Symposium on Operating Systems Design & Implementation, OSDI'10, 2010.
24. Neumeyer L, Robbins B, Nair A, and Kesari A. S4: Distributed stream

computing platform. In: Proceedings of 2010 IEEE International Conference on Data Mining Workshops, ICDMW '10, 2010, 170–177.
25. Gropp W, Lusk E, and Skjellum A. Using MPI: portable parallel programming with the message-passing interface. MIT Press, 1994.
26. MPICH2 : High-performance and widely portable MPI. http://www.mcs.anl.gov/research/projects/mpich2/.
27. Graham R L, Shipman G M, Barrett B, Castain R H, Bosilca G, and Lumsdaine A. Open MPI: a High-Performance, heterogeneous MPI. In: Proceedings of 2006 IEEE International Conference on Cluster Computing, Cluster '06, 2006.
28. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, and Zaharia M. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.
29. Wentzlaff D, Gruenwald III C, Beckmann N, Modzelewski K, Belay A, Youseff L, Miller J, and Agarwal A. An operating system for multicore and clouds: mechanisms and implementation. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, 2010, 3–14.
30. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, and Stoica I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European conference on Computer systems, EuroSys '10, 2010, 265–278.
31. Benson T, Akella A, and Maltz D A. Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th annual conference on Internet measurement, IMC '10, 2010, 267–280.
32. Boutaba R, Cheng L, and Zhang Q. On cloud computational models and the heterogeneity challenge. Journal of Internet Services and Applications, 2012, 3(1):77–86.
33. Zaharia M, Konwinski A, Joseph A D, Katz R, and Stoica I. Improving MapReduce performance in heterogeneous environments. In: Proceedings of the 8th USENIX conference on Operating Systems Design & Implementation, OSDI '08, 2008.
34. Fan Z, Qiu F, Kaufman A, and Yoakum-Stover S. GPU cluster for high performance computing. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC '04, 2004.
35. Liu J, Chandrasekaran B, Wu J, Jiang W, Kini S, Yu W, Buntinas D, Wyckoff P, and Panda D K. Performance comparison of MPI implementations over InfiniBand, myrinet and quadrics. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, SC '03, 2003.
36. Greenberg A, Hamilton J, Maltz D A, and Patel P. The cost of a cloud: research problems in data center networks. ACM SIGCOMM Computer Communication Review, 2008, 39(1):68–73.
37. Abadi D J. Data management in the cloud: Limitations and opportunities. IEEE Data Engineering Bulletin, 2009, 32(1):3–12.
38. Buyya R, Beloglazov A, and Abawajy J H. Energy-Efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '10, 2010, 6–20.
39. Ramgovind S, Eloff M M, and Smith E. The management of security in cloud computing. In: Proceedings of the 9th Annual Information Security for South Africa Conference, ISSA '10, 2010.
40. Mehta S, and Neogi A. ReCon: a tool to recommend dynamic server consolidation in multi-cluster data centers. In: Proceedings of the 11th IEEE/IFIP Network Operations and Management Symposium, NOMS '08, 2008, 363–370.
41. Zhan J, Wang L, Tu B, Li Y, Wang P, Zhou W, and Meng D. Phoenix cloud: Consolidating different computing loads on shared cluster system for large organization. In: Proceedings of the 1st Workshop on Cloud Computing and its Applications, CCA '08, 2008.
42. Calheiros R N, Ranjan R, Beloglazov A, De Rose C A F, and Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 2011, 41(1):23–50.
43. Livny M. Condor and the cloud - the challenges and the roadmap of condor. http://www.grid.org.il/_Uploads/dbsAttachedFiles/Condor-Cloud-IGT.pdf, 2009.
44. Linux containers. http://lxc.sourceforge.net/.
45. Koziolek H. Performance evaluation of component-based software systems: A survey. Performance Evaluation, 2010, 67(8):634–658.
46. Huai Y, Lee R, Zhang S, Xia C H, and Zhang X. DOT: a matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, SoCC '11, 2011, 1–14.
47. Thain D, Tannenbaum T, and Livny M. Distributed computing in practice: the condor experience. Concurrency and Computation: Practice and Experience, 2005, 17(2-4):323–356.
48. Youseff L, Butrico M, and Da Silva D. Toward a unified ontology of cloud computing. In: Proceedings of Grid Computing Environments Workshop, GCE '08, 2008.
49. Apache mesos: Dynamic resource sharing for clusters. http://incubator.apache.org/mesos/.
50. Lee G, Chun B, and Katz R H. Heterogeneity-Aware resource allocation and scheduling in the cloud. In: Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud '11, 2011.
51. Zaharia M, Chowdhury M, Franklin M J, Shenker S, and Stoica I. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, 2010.
52. Apache ZooKeeper. http://zookeeper.apache.org/.
53. Murthy A C. The next generation of apache hadoop MapReduce. http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/, 2011.
54. Apache HBase. http://hbase.apache.org/.
55. Seo S, Yoon E J, Kim J, Jin S, Kim J, and Maeng S. HAMA: an efficient matrix computation with the MapReduce framework. In: Proceedings of the 2nd International Conference on Cloud Computing Technology and Science, CloudCom '10, 2010, 721–726.
56. Apache giraph. http://incubator.apache.org/giraph/.
57. Pandey J. RPC improvements and wire compatibility in apache hadoop. http://hortonworks.com/blog/rpc-improvements-and-wire-compatibility-in-apache-hadoop/, 2012.
58. Wright D. Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with condor. In:

Proceedings of the LCI International Conference on Linux Clusters: The HPC Revolution, 2001.
59. Thain G. Condor integrated with hadoop's map reduce. http://research.cs.wisc.edu/condor/CondorWeek2010/condor-presentations/thain-condor-hadoop.pdf, 2010.
60. Foster I, and Kesselman C. Globus: A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications, 1997, 11(2):115–128.
61. Henderson R. Job scheduling under the portable batch system. In: Feitelson D, and Rudolph L, editors, Job Scheduling Strategies for Parallel Processing, volume 949 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, 279–294.
62. Coleman N, Raman R, Livny M, and Solomon M. Distributed policy management and comprehension with classified advertisements. Technical Report UW-CS-TR-1481, Computer Sciences Department, University of Wisconsin-Madison, 2003.
63. Couvares P, Kosar T, Roy A, Weber J, and Wenger K. Workflow management in condor. In: Taylor I J, Deelman E, Gannon D B, and Shields M, editors, Workflows for e-Science. Springer London, 2007, 357–375.
64. Basney J, and Livny M. Deploying a high throughput computing cluster. In: Buyya R, editor, High Performance Cluster Computing: Architectures and Systems, volume 1. Prentice Hall PTR, 1999, 116–134.
65. Farrellee M. Condor: Cloud scheduler. http://spinningmatt.files.wordpress.com/2010/04/ matthewfarrelleeopensourcecloudcomputingforum10feb2010.pdf, 2010.
66. Open grid scheduler: The official open source grid engine. http://gridscheduler.sourceforge.net/.
67. Son of grid engine. https://arc.liv.ac.uk/trac/SGE.
68. Sun Microsystems. Sun ONE grid engine, enterprise edition administration and user's guide. Technical Report 816-4739-11, 2002.
69. Troger P, Rajic H, Haas A, and Domagalski P. Standardization of an API for distributed resource management systems. In: Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, CCGRID '07, 2007, 619–626.
70. Gentzsch W. Sun grid engine: towards creating a compute power grid. In: Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGIRD '01, 2001, 35–36.
71. Oracle Corporation. Extreme scalability using oracle grid engine software: Managing extreme workloads. Technical report, 2010.
72. Templeton D. Intro to service domain manager. http://blogs.oracle.com/templedf/entry/service_domain_manager, 2010.
73. Sotomayor B, Montero R S, Llorente I M, and Foster I. Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing, 2009, 13(5):14–22.
74. Mugler J, Naughton T, and Scott S L. OSCAR meta-package system. In: Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications, HPCS '05, 2005, 353–360.
75. Massie M L, Chun B N, and Culler D E. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, 2004, 30(7):817–840.
76. Zha L, Li W, Yu H, Xie X, Xiao N, and Xu Z. System software for china national grid. In: Proceedings of IFIP International Conference on Network and Parallel Computing, NPC '05, 2005, 14–21.
77. Lin J, Lu X, Yu L, Zou Y, and Zha L. VegaWarden: a uniform user management system for cloud applications. In: Proceedings of the 5th IEEE International Conference on Networking, Architecture and Storage, NAS '10, 2010, 457–464.
78. Yu L, Zha L, Wang X, Zhou H, and Zou Y. GOS security: Design and implementation. In: Proceedings of the 15th International Conference on Parallel and Distributed Systems, ICPADS '09, 2009, 955–960.
79. Steinder M, Whalley I, Carrera D, Gaweda I, and Chess D. Server virtualization in autonomic management of heterogeneous workloads. In: Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, IM '07, 2007, 139–148.
80. Mateescu G, Gentzsch W, and Ribbens C J. Hybrid Computing-Where HPC meets grid and cloud computing. Future Generation Computer Systems, 2011, 27(5):440–453.

Jian Lin is a PhD candidate in computer architecture at Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include distributed software architecture, large-scale resource management, and security technologies in grid and cloud computing systems.

Li Zha obtained his PhD degree in 2003, and is an associate professor of Institute of Computing Technology, Chinese Academy of Sciences. He was been the project leader of several national level research programs. His research is focused on large-scale distributed resource management, data storage/processing/retrieval and system level optimization. His interests also include other classic issues in distributed computing and grid computing field.

Zhiwei Xu received the PhD degree from University of Southern California in 1987. He is currently a professor of Institute of Computing Technology, Chinese Academy of Sciences. His research interests include network computing, distributed operating systems, and high-performance computer architecture. His editorial board services include the *IEEE Transactions on Services Computing*, *Journal of Grid Computing*, *Journal of Computer Science and Technology*, and *Journal of Computer Research and Development*. He is a senior member of the IEEE.