Enabling Distributed Computing Systems with elop^{TM}

Junwei Cao Shuo Chen Yuxin Wan Wei Chen
Research Institute of Information Technology
Tsinghua National Laboratory for Information Science and Technology
Tsinghua University, Beijing 100084, P. R. China
e-mail: jcao@tsinghua.edu.cn

Abstract— **In the era of Internet, one of the most ultimate goals is that users can seamlessly access to any computing application or service at anywhere anytime. Computing infrastructuralization is the path to gradually address the challenge. In this work, elop^{TM} computing is proposed that encapsulates and integrates various computing elements as services, benefiting from existing advanced computing technology, e.g. virtualization and SaaS. Higher levels of services are also provided, e.g. metadata management, resource management and scheduling, security and authorization, which can be utilized to build distributed computing systems in a scalable way. In this paper, the architecture of elop^{TM} computing is illustrated and corresponding middleware implementation is introduced in details. A typical scenario is given to demonstrate the potential of enabling distributed computing systems with elop^{TM}.**

Keywords- ***Distributed Computing; Computing Infrastructuralization; elop^{TM} computing; Grid Computing; Cloud Computing; Virtual Organizations; Internet of Things***

I. INTRODUCTION

With several decades of development of distributed computing and Internet technology, the whole IT industry has evolved into an era of service-oriented and human-centric computing. One of the most ultimate goals is that users can seamlessly access to applications or services regardless of locations, time and devices, which is the so called A4 (Anytime Anywhere Any Application) issue. While many existing computing paradigms, e.g. grid computing [13] and cloud computing [2], can be utilized to partially address the A4 challenge, new computing architecture is required to provide an uniform framework from a perspective of *computing infrastructuralization*.

Infrastructure is basic physical and organizational structures needed for the operation of a society or enterprise, or the services and facilities necessary for an economy to function [15]. Typical infrastructures include transportation systems, power grids [12], telecommunication and Internet. Well-designed infrastructure supports the distribution of services in a safe, reliable, convenient and affordable way. All the society supporting technology comes to infrastructure at the end and the corresponding process is called *infrastructuralization*. To enable knowledge economy in the 21st century, new infrastructure requirements arise stimulated by dramatic development of information technology, especially distributed computing and communication technology. This is so-called *cyberinfrastructure* (CI) [5][17], proposed by the US National Science Foundation (NSF) as the fundamental driving force of science and technology innovation [3].

Resource infrastructuralization in cyberspace is becoming an inevitable trend, with the development of grid and cloud computing. While grid computing is focused more on cross-domain resource sharing, the applications and services are not able to be distributed efficiently. Main applications of grid computing are high performance computing and massive data management in scientific research areas [1]. However, most of requirements for computing services are massive, various, small-scale services. These services should be accessed anywhere, anywhere and regardless of types of devices. Cloud computing is focused on service provisioning but back-bone data centers cannot scale well without consideration of resource infrastructuralization [18]. These challenges have to be addressed by new computing infrastructure that can provide an uniform framework for both resource infrastructuralization and service provisioning. In this work, we propose *elop^{TM} computing* from a perspective of computing infrastructuralization.

One of the most fundamental principles of infrastructure is to standardize and encapsulate heterogeneous elements that can be dynamically aggregated according to different requirements. In this work, we propose elop^{TM} computing architecture, on which heterogeneous computing resources are organized and shared for various applications. Computing resources are separately described and encapsulated instead of using a traditional all-in-one model. These include computing hosts (physical or virtual machines [4]), data and storage [7], software packages [11] and displays (remote displays or virtual desktops [16]). To solve the A4 problem, the physical properties of computing elements are totally abstracted and described in a logical way. Resource management and scheduling can be also fulfilled at the logical layer [10]. In addition, dynamic construction of virtual organizations (VO) is supported for security, trust and privacy management of various computing elements [8]. At a higher level, process management is provided to enable aggregated service provisioning and application development [9], which is similar to workflow management for grid computing [6].

In Section II, the principle and architecture of elop^{TM} computing is illustrated. In section III, a reference software implementation is given. Section IV demonstrates an example application to show how we solve the A4 problem and outline future application scenarios. The paper is concluded in Section V.

II. PRINCIPLES AND ARCHITECTURE

The design of elop^{TM} architecture is motivated by infrastructuralization of computing services.

The main objective of elop™ computing is to integrate heterogeneous computing resources as a new infrastructure to provide computing capacity as services. The progress of distributed computing has brought about valuable technology to achieve this goal such as grid computing and cloud computing. Grid computing is designed to integrate services across distributed, heterogeneous, dynamic virtual organizations formed from the disparate resources to serve large-scale applications such as e-business and e-science. It is application oriented for tightly coupled, high performance computing. However, when it comes to less coupled, massive and transaction based requests, grid computing is not a proper solution. Cloud computing developed in recent years evolves to be an option but is not focused on scalable resource management.

elop™ computing brings out a new perspective to decouple essential processes of computing services into various elements, e.g. computing, storage, software and display. When these elements are physically encapsulated and logically abstracted, the process of computing services becomes a matter of on-demand dynamic reorganization of these elements. For example, we consider movie watching as a typical scenario. Virtual machines are provided to host the service. Data management is responsible for storing movie files and transmitting to the service host. The software service manages decoding software to be installed on the service host. The decoded video is streaming over the Internet and displayed anywhere. The kernel of elop™ computing is to manage and connect logical computing elements and to develop platforms and interfaces for heterogeneous resources allocation.

The concrete architecture of elop™ computing is shown in Figure 1, including four layers (elements, logics, organizations, and processes). The functions of these layers are described as follows.

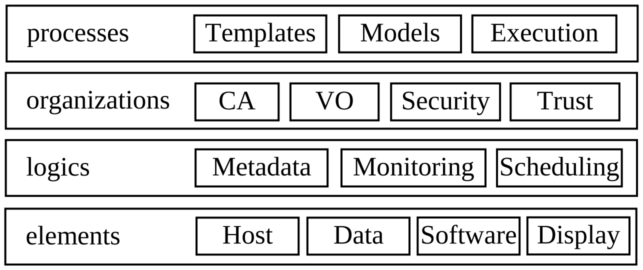

Figure 1. elop™ Architecture.

- e(lement): This is the layer of elop™ computing. In traditional all-in-one model, all data and software are managed on one host with local display. For elop™ computing, computing elements are separately encapsulated and managed in a distributed way. Elements offer interfaces to share specific description of physical properties and dynamic resource status and manage requests and return results. There could be more elements according to different scenarios. For cloud computing, the *host* can be extended to *vmhost* (virtual machine hosts).
- l(ogic): Physical properties of elements are abstracted to logical description. Functions and interfaces of elements are isolated from specific locations and extended to the overlay of Internet. This provides an uniform interface for element access, with optional additional supports, e.g. resource scheduling. In this way, applications do not need to care about details of basic elements such as their locations and status.
- o(rganization): The logical elements are virtually organized in this layer. It offers several mechanism including CA (certificate authority), VO (virtual organization) to guarantee access security, authorization, and management of sharing.
- p(rocess): This is the application enabling layer of elop™ computing. It offers interfaces to users to operate and allocate elements for a given application, to supervise the status of the elements, and to manage authorities. It also provides general tools for application process template construction, process modeling and runtime environments.

In this architecture, we do not put any extra limitation to elements except the specification of interfaces and protocol of communication. Therefore, these elements can be logically reallocated and described. The security mechanism is built upon the description of elements and the authorization of users.

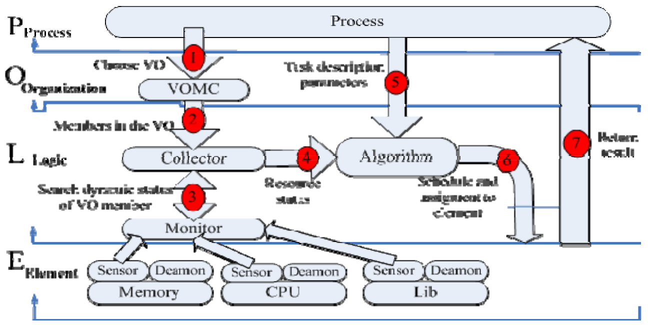

Figure 2. Job Scheduling with elop™.

Job scheduling is a typical scenario in traditional parallel and distributed computing. The process of a job that a user assigns to an elop™ system is as follows:

1) The user chooses a VO to execute the job.
2) The job information and the certificate of the user are submitted to the logic layer. A job *collector* receives job requests and put them in a job queue.
3) At the logic layer, the *monitor* receives the element list of the VO (resources sharing and user authorization are enabled only within a VO) and checks the status of elements. Each element has a managing daemon or sensor running, submitting dynamic status information to the logic layer periodically.
4) The *collector* sends all element information to the scheduler that is pre-defined with a scheduling algorithm or strategy.

5) The *scheduler* collects both information of resources and jobs and generates job execution schedules.
6) The job is allocated to the targeting elements according to the schedule.
7) The results are finally returned to the process layer.

III. A REFERENCE IMPLEMENTATION

According to the new architecture proposed above, we started to provide a reference implementation at Tsinghua University in 2010. Supporting technologies for elop^{TM} implementation are listed below.

- CA and digital signatures for identity authentication.
- Encrypted communication based on sockets and X.509 certification[1].
- Virtual machine management and elastic allocation based on VirtualBox[2] and KVM[3].
- The RDP protocol[4] implementation using rdesktop[5] for virtual desktop implementation.
- Database management and access based on Unix ODBC[6].
- Web services implementation using gSAOP[7].

The reference implementation is developed in the C++ language and compiled on the Linux platform. Every layer of the elop^{TM} architecture has its standalone software package that can be separately installed on the different Linux servers. Once these servers are interrelated by the application process, they are virtually organized and required services can be accessed via the overlay of network anywhere and anytime.

Any instance including users and stand-alone services has its unique certificate to represent its identification. The functions of the stand-alone servers in each layer are listed as follows.

1) The CI of the organization layer
The CI maintains three database tables. One records VO's names and their description, the second records relationships between VOs and users or elements, the third records the certificate and corresponding authorization.
Functions:
A. CI manages all the element information including its name, certificate, description, and type (a user or element). All the element servers contact with CI when started.
B. CI accepts application for VO set up from users and creates VO tables according to details of application including VO name, VO description, and VO's certificate name.
C. CI accepts requests for listing all VO names from users.
D. CI accepts requests for listing all VO names that an user belongs to.
E. CI accepts requests for deleting VOs if the corresponding sender is the administrator of the VO.
F. CI accepts requests for adding or removing elements into or from VOs, and the administrator of a VO decides whether the element can be accepted into or removed from a VO.
G. CI accepts requests for adding or deleting users into or from VOs, and the administrator decides whether the user can be accepted into or removed from a VO.

2) VOs of the organization layer
A VO maintains two database tables. One records all the information of elements including certificates in the VO, the other records the authorization of the elements and users.
Functions:
A. A VO connects to the CI when started and submits its IP address and port number to CI. The information is transmitted cryptographically by CA public key. In the connection process, VO will submit the VO certificate. CI will judge whether the submitted certificate matches the certificate of VO when it is set up.
B. VO accepts requests from users for joining. But the authorization of the user is null.
C. VO accepts requests for listing all the users' authorization.
D. VO accepts requests for listing all the elements' information.
E. VO accepts requests for removing elements from the administrator and reports updates to the CI.
F. VO accepts requests for removing users from the administrator and reports updates to the CI.
G. VO accepts requests from the administrator for revising authorizations of users in the VO.
H. VO accepts requests from the CI for adding elements. VO checks the user's authorization whether it can add elements. If the user has authorization, VO will add this element and return results to the CI.
I. VO receives authorization verification requests from logic servers.

3) Logics of the logic layer
Logic servers maintain two tables in general. One records dynamic information and the other records element-specific information.
Functions:
A. Logic receives requests from applications and users. The user should assign a certain VO first, and then the logic server applies VO for authorization verification for each request. The request includes user's certificate.
B. Logic acquires the dynamic status of elements periodically.
C. Logic receives static description of elements.

4) Elements of the element layer
Functions:
A. Element connects to the CI when started and submits the name and certificate to the CI.

[1] http://www.ietf.org/rfc/rfc2459
[2] https://www.virtualbox.org/
[3] http://www.linux-kvm.org/
[4] http://en.wikipedia.org/wiki/Remote_Desktop_Protocol
[5] http://www.rdesktop.org/
[6] http://www.unixodbc.org/
[7] http://www.cs.fsu.edu/~engelen/soap.html

B. Element exchanges information with logics.
C. Element receives job requests from logics.
D. Element returns results directly to users.

Some basic computing elements have been developed in the reference implementation. These include: *host*, *vmhost*, *database*, *software* and *display*.

- The element *host* is used to encapsulate a physical server. Dynamic information, e.g. CPU frequency, RAM memory size and hard disk size, are monitored and submitted to logics.
- The element *vmhost* is inherited from *host*. It encapsulates a physical server that operates multiple virtual machines. Current implementation interfaces with different virtualization technologies and corresponding management tools, e.g. VirtualBox and KVM.
- The element *database* is a specific inheritance of a general element *data*. It provides uniform interfaces for database access. Other specific data elements include *file*, *xml*, and *sensor*, which are under development.
- The element *software* provides an encapsulation of existing software repositories. It provides uniform interfaces for software package management tools, e.g. YUM. It also provides automatic tools for installing software into virtual machines.
- The element *display* is an encapsulation of existing display servers. For example, an iPad can be considered as a display server that provides display services to users. The display implementation provides uniform interfaces to existing display access tools, e.g. rdesktop, since most existing host servers, e.g. Windows and VirtualBox, support the RDP protocol.

Besides layered implementation described above, some common utilities have also to be implemented, which are discussed in details below.

1) Communication

Both synchronous and asynchronous communication are supported via sockets. Web service interfaces are implemented using the C/C++ web service tool gSoap. Asynchronous communication is utilized for interactions between components of different layers among the elop^{TM} architecture. Synchronous communication is mainly used for users to call for elop^{TM} functionalities.

2) Security

The communication of the reference implementation is built on X.509 encrypted socket communication which is a strict hierarchical system of certificate authorities (CA) for issuing the certificates. Thus, the unique identification for any instance in a runtime system is set as the name of the certificate. Requests and returns of every standalone server are classified into two types, information and data.

3) Authorization

Authorization is managed via VOs. Users can set up their own VO. In a VO, the owner can add and remove elements and users. Users have authorization to operate the elements in the same VO for resources sharing. A user can register to many different VOs for different tasks. An element can also belong to many VOs for resource sharing. There is only one CI in the whole environment issuing certificates but there could be many VOs. Elements and users can move among different VOs, supporting dynamic VO management.

4) Database

Each layer has to maintain databases for information management. All elements keep a record in CI, which issues identifications for them. Each VO manages information of elements and users belonging to it. Each logic manages metadata for both dynamic and static information of registered elements. Scalable system implementation can be achieved in this way, since not all interactions have to go through a center, which otherwise may become a bottleneck.

5) APIs

The elop^{TM} kernel is only a framework for better supporting distributed application development. Abundant application programming interfaces (APIs) are provided. Basic classes for every layer are developed and typical elements, logics, VOs and the CI are implemented. System developers just need to focus on application-specific issues and inherit from basic classes of elop^{TM} so as to benefit from existing management that elop^{TM} already provides.

IV. A CASE STUDY

In this section, a demonstration is provided as an example system implementation using elop^{TM}. A user can watch a movie through a mobile device which is connected to a remote desktop of a virtual machine running on a cluster managed by elop^{TM}. The scheme of this application is shown in Figure 3.

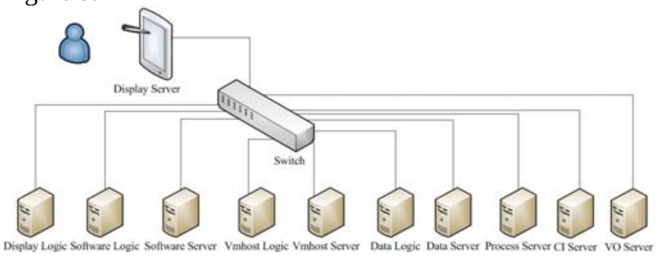

Figure 3. A elop^{TM} System Implementation

Different components of elop^{TM} is deployed separately on a CentOS Linux server cluster. The display server that can open remote desktop of a virtual machine runs on the ARM platform with an embedded Linux system. All the devices are connected to an Ethernet switch.

From the elop^{TM} point of view, the scenario above forms a complete functional cluster, since all necessary layers of elop^{TM} are connected through networks. The CI server records and manages all elments. The VO server supervises to identify authorization of virtual organization. All logic servers screen physical properties of its elements, transfer and schedule instructions from the process layer. In this way, the system can use a given software (video decoding in this case), operate on given data (movies in this case), run it on a given host (virtual machine hosts in this case), and display it on a given terminal (mobile devices in this case).

The whole process of our experiment is scheduled on the process server. It firstly opens the virtual machine on the *vmhost*. Then it asks the element of *data* for the required movie data file. The element of *software* has to provide video decoder software for the virtual machine and to install it and start the decoder to play the movie by SSH. Then, the user can watch the movie through the *display* server. In this case study, we somehow decouple basic computing elements (hosts, data, software, and display) and manage them as networked services.

The logics are also playing important roles in this scenario. There could be multiple available virtual machines that can host the movie watching service. How to select among multiple resources is determined by the vmhost logic. The required software and data could also be provided by many repositories. There could be even multiple display servers and users may require display as a service. This can be easily handled by choosing a proper display by the display logic.

The elop^{TM} architecture can be utilized for dynamic aggregation of basic computing elements for a given distributed application with users' requirements. It is a middleware and toolkit that can be utilized to provide basic supports (e.g. communication, security, authorization, databases, and APIs) for development of similar distributed computing systems.

V. CONCLUSIONS

In this paper, we first introduce the concept of infrastructure and point out computing infrastructuralization becomes an inevitable trend. Thus, we propose the elop^{TM} computing architecture and demonstrate how it works with a typical case study. Heterogeneous computing resources are decoupled and encapsulated as networked services, which can be reorganized for a given application in a on-demand way. The elop^{TM} architecture features dynamic virtual organization management for authentication and authorization control, in a similar way to social networks.

More specific computing elements are under-development to incorporate other heterogeneous resources, e.g. sensors. Besides, elop^{TM} reference implementation is extended to both embedded systems and server environments. With C/C++ implementation, the reference implementation can achieve high performance on both embedded systems and high performance cluster systems. Most cloud applications require terminal access to massive resources on the cloud and elop^{TM} can be used as a basis to provide a uniform solution for cloud computing [14].

More application solutions based on the elop^{TM} architecture are under-development for various distributed scenarios, e.g. Internet of Things (IoT)[8]. The element *data* can be extended to an element *sensor*, supporting continuous data streaming [10]. In general, sensors play embedded between cyber and physical spaces, enabling distributed computing for IoT applications [14].

ACKNOWLEDGEMENT

This work is supported by Ministry of Science and Technology of China under National 973 Basic Research Program (grant No. 2011CB302805) and National 863 High-Tech Program (grant No. 2011AA040501).

Contributors of elop^{TM} development also include Zhen Wang, Xiao Lin, Ziyang Li, Han Qian, Qin Lv, Yayu Ni, Yan Huo, Yunhao Li, Fei Wang, Lei Jia, Jinliang Huang, Ye Chen, Yaping Gu, and Miao Wang.

REFERENCES

[1] B. Allcock, A. Chervenak, I. Foster, C. Kesselman, M. Livny, "Data Grid Tools: Enabling Science on Big Distributed Data", *J. Physics: Conference Series*, 16, 571-575, 2005.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, et, al, "Above the clouds: A Berkeley View of Cloud Computing", *Technical Report No. UCB/EECS-2009-28, University of California, Berkerley*, 2009.

[3] D. E. Atkins, et al., "Revolutionizing Science and Engineering through Cyberinfrastructure", *National Science Foundation Blue – Ribbon Advisory Panel on Cyberinfrastructure*, January 2003.

[4] P. Barham, B. Dragovic, K. Fraser, et. al., "Xen and the Art of Virtualization", *Proc. ACM Symp. on Operating Systems Principles*, 2003.

[5] J. Cao (Ed.), *Cyberinfrastructure Technologies and Applications*, Nova Science Publishers, 2009.

[6] J. Cao, S. A. Jarvis, S. Saini and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing", *Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, Tokyo, Japan, 198-205, 2003.

[7] J. Cao and J. Li, "Large-scale Real-time Data-driven Scientific Applications", *Proc. 2nd Int. Conf. on Networking and Distributed Computing*, Beijing, China, 116-121, 2011.

[8] J. Cao and Z. Wang, "VOMES: a Virtual Organization Membership Evaluation System", *Int. J. Networking and Virtual Organisations*, 10(1), 88-108, 2012.

[9] J. Cao, F. Zhang, K. Xu, L. Liu, and C. Wu, "Formal Verification of Temporal Properties for Reduced Overhead in Grid Scientific Workflows", *J. Computer Science and Technology*, 26(6), 1017-1030, 2011.

[10] J. Cao, W. Zhang and W. Tan, "Dynamic Control of Data Streaming and Processing in a Virtualized Environment", *IEEE Trans. Automation Science and Engineering*, 9(2), 365-376, 2012.

[11] W. Chen, J. Cao, and Z. Li, "Customized Virtual Machines for Software Provisioning in Scientific Clouds", *Proc. 2nd Int. Conf. on Networking and Distributed Computing*, Beijing, China, 240-243, 2011.

[12] M. Chetty and R. Buyya, "Weaving Computational Grids: How Analogous are they with Electrical Grids?", *IEEE Computing in Science and Engineering*, Vol. 4, No. 4, pp. 61-71, 2002.

[13] I. Foster and C. Kesselman (Eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufman, 1999.

[14] K. Hwang, G. C. Fox, and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2012.

[15] Infrastructure from Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Infrastructure.

[16] X. Liao, H. Jin, L. Hu, H. Liu, "Towards Virtualized Desktop Environment", *Concurrency and Computation: Practice and Experience*, Vol. 22, No. 4, pp. 419–440, 2010.

[17] J. Yin, J. Cao, Y. Wang, L. Liu, and C. Wu, "Scheduling Remote Access to Scientific Instruments in Cyberinfrastructure for Education and Research", *Proc. 7th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, 426-433, 2007.

[18] F. Zhang, J. Cao, C. Hong, J. J. Mulcahy, and C. Wu, "Provisioning Virtual Resources Adaptively in Elastic Compute Cloud Platforms", *Int. J. Web Services Research*, 8(3), 54-69, 2011.

[8] http://en.wikipedia.org/wiki/Internet_of_Things