Journal of Computer Science and Technology

Indexed in: SCIE, EI, Scopus, INSPEC, DBLP,CSCD, etc.
Sponsored by: ICT, CAS & CCF

JCST Papers

Only for Academic and Non-Commercial Use

Thanks for Reading!

JCST WeChat Subscription Account

JCST URL: https://jcst.ict.ac.cn

SPRINGER URL: https://www.springer.com/journal/11390

E-mail: jcst@ict.ac.cn

Online Submission: https://mc03.manuscriptcentral.com/jcst

Twitter: JCST_Journal

LinkedIn: Journal of Computer Science and Technology

Xu ZW, Yu ZS, Li FZ *et al.* Hypertasking: From information web to computing utility. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY, 40(3): 607-620, May 2025. DOI: 10.1007/s11390-025-5126-4, CSTR: 32374.14.s11390-025-5126-4

Hypertasking: From Information Web to Computing Utility

Zhi-Wei Xu (徐志伟), *Fellow, CCF*, Zi-Shu Yu (俞子舒), *Student Member, CCF*
Feng-Zhi Li (李奉治), *Student Member, CCF*, and Yao Zhang (张　垚), *Student Member, CCF*

Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 100149, China

E-mail: zxu@ict.ac.cn; yuzishu19s@ict.ac.cn; lifengzhi20z@ict.ac.cn; zhangyao22z@ict.ac.cn

Received December 26, 2024; accepted February 11, 2025.

Abstract John McCarthy proposed the vision of utility computing in 1961. Barbara Liskov proposed a related vision of abstraction-powered Internet Computer in 2009. This position paper outlines a distributed computing model towards realizing the McCarthy-Liskov vision. This "hypertasking" model aims at extending the "hypermedia" model of the World Wide Web into a model of World Wide Computing Utility, turning an information web into a computing web. The hypertasking model contains three abstractions, including global resource space, stored-computer architecture, and monadic hypermedia. A prototype architecture and experimental evidence are presented to support this perspective.

Keywords cloud computing, computing utility, hypermedia, hypertask, stored-computer architecture

1 Introduction

The idea of utility computing was proposed over 60 years ago by John McCarthy[1] in his 1961 lecture to celebrate MIT's centennial. He envisioned that "computation may someday be organized as a public utility, just as the telephone system is a public utility … The computing utility could become the basis for a new and important industry".

Today's cloud computing systems and applications have only partially realized McCarthy's computing utility vision[1]. Cloud computing has become an important industry, with increasing market scale. However, it is still far from a convenient public utility, compared with the universal ease offered by telephony. Using a cell phone, one can instantly reach any other phone on the planet, although the two phones may be operated by different telecommunication service providers. In contrast, a cloud computing user is tethered to a specific cloud provider at any given time.

McCarthy's lecture also offered intellectual insights. He considered the utility computing concept as fundamentally important as the time-sharing concept and the stored-program concept. He also mentioned three salient features of the computing utility: 1) pay-per-use ("each subscriber needs to pay only for the capacity he actually uses"), 2) large-computer culture ("but he has access to all programming languages characteristic of a very large system"), and 3) private computer ("a computer that he can have continuously at his beck").

Almost half a century later, Barbara Liskov made intellectual revisions to the computing utility concept. In her 2009 Turing award lecture, Barbara Liskov envisioned "Internet as a Computer" as a future research direction[2]. Liskov emphasized that this Internet Computer should be abstraction powered. She lamented on the status of distributed systems: "There is a funny disconnect how we write distributed programs. You write individual modules. But then, when you want to connect them together, you are out of the programming language and sort of into this other world. Maybe we need languages that are a little bit more complete now, so that we can write the whole thing in the language."

Perspective
Special Section of CCF Computility 2024
The work was supported by the National Natural Science Foundation of China under Grant Nos. 62072434 and U23B2004.

It is probably not a coincidence that McCarthy served as Liskov's PhD thesis adviser. Respecting this academic lineage, we call the joined vision of computing utility and Internet Computer the McCarthy-Liskov vision. Such a computing utility should have the following salient features.

- *Pay-per-Use Services.* Users subscribe to computing services of the utility and pay only for resources actually used.
- *Planet-Scale Culture.* Users have ready access to all computer culture of one worldwide utility from anywhere at any time.
- *Low-Entropy Systems.* Each of the billions of worldwide subscribers sees a private computer, largely isolated from disorders such as error propagation, workload interferences, and system jitters.
- *Abstraction-Powered Programs.* Developing and running programs on the computing utility should be powered by abstractions, especially programming language abstractions.

Our team of Information Superbahn research added one more "high-goodput" performance requirement for computing utility[3, 4].

- *High-Goodput Utility.* The computing utility is an efficient system, such that most executed tasks show good enough user experiences.

The World Wide Web (WWW) functions as an information utility, wherein any person or agent (producer) can put up a piece of information on the Web as a webpage. Consequently, anyone (a consumer or a user) in the world can instantly access and utilize it. In fact, modern cloud computing has benefited from the WWW technology. One of the first commercial public cloud computing offerings, Amazon cloud computing services, is called Amazon Web Services (AWS).

The WWW as an industry and an ecosystem is supported by the concept of worldwide hypermedia, as well as abstractions like uniform resource identifier (URI), HTTP, and HTML. Partly because the Web is abstraction-powered, it has the following ease of use property.

- *One-Click Instant Access.* By a single click on a hyperlink, a user of the Web can instantly access a target webpage, which can be an information resource anywhere in the world.

This ease-of-use property implies low runtime latency and low human effort. Many webpages on today's Web can be accessed with runtime latencies within a second or even milliseconds. A user sees an information resource provided in a webpage, visible to the user as a hyperlink or a link word. Many invisible details relevant to the access are handled by abstractions and their corresponding runtime software.

However, the WWW today is not yet an Internet Computer. The WWW is an "information" web, rather than a "computing" web or a computing utility.

One way to realize the McCarthy-Liskov vision is to advance from the World Wide Web to a World Wide Computing Utility. We need to stand on the shoulders of giants, such as:

- McCarthy's computing utility concept with the pay-per-use, large, private computer idea[1],
- Liskov's idea of the abstraction-powered Internet Computer[2],
- Tim Berners-Lee's idea of hyperlinking the world via the WWW technology[5], and
- communities efforts in grid computing[6, 7], cloud computing[8, 9], and sky computing[7, 10–13], as discussed in Section 4.

Innovations are needed to extend the hypermedia model of the WWW into a hypertask model that accommodates additional requirements and challenges of the computing utility, thus turning an information web into a computing web.

The rest of this paper is organized as follows. Section 2 highlights the main characteristics of the hypertasking model of computing utility, in contrast to the hypermedia model of the WWW. Section 3 provides supporting evidence by presenting a prototype architecture and experimental evaluation. Section 4 discusses related work. Section 5 offers concluding remarks.

2 Hypertask Compared with Hypermedia

The hypertask concept was first proposed in the context of the Internet of Things (IoT) and edge computing[14, 15]. In this section, we highlight the more fundamental nature of hypertasking as a computing model that aims at extending the WWW into a World Wide Computing Utility, realizing the McCarthy-Liskov vision.

The familiar hypermedia model is shown in Fig.1(a). The WWW provides a uniform resource space where any resource has a URI. A user can access a resource from a client device, such as a laptop computer, by entering a URI in a web browser and receiving a webpage. The webpage in Fig.1(a) displays three types of hyperlinks to web resources, each

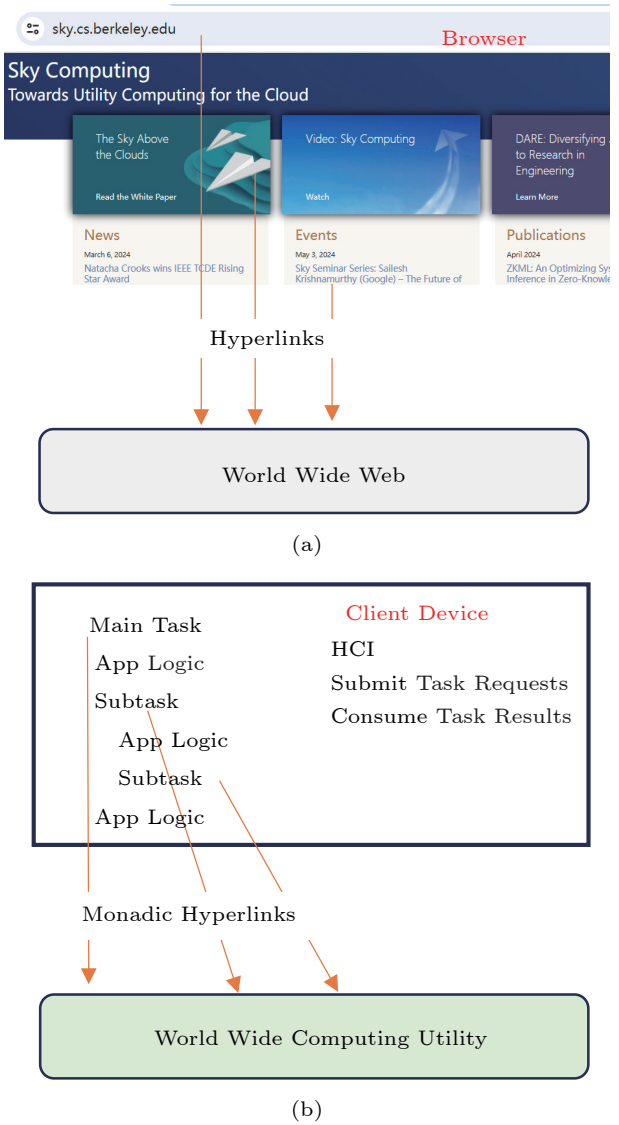

Fig.1. Hypertasking versus hypertexting. (a) Hypermedia model for WWW. (b) Hypertasking model for computing utility.

accessible with just one click. The URI points to the homepage of the UC Berkeley Sky Computing Lab①. The hyperlink for the picture resource is an embedded link, which the browser will access automatically. The seminar event hyperlink requires a click.

Fig.1(b) shows the hypertasking computing model and its three principles. We use the word "computility"[16] as a shorthand for "world wide computing utility", and assume there are many client devices but only one computility when explaining Fig.1(b).

A hypertask consists of application logic (app) code and systems code, and is organized as a main task and zero or more subtasks. Each task uses one or more resources in the computility via one-click monadic hyperlinks. A hypertask may also serve as a subtask of another. The main tasks and subtasks are also hypertasks.

Visible to a client device, each hypertask is executed by a Request-Response-Consume (RRC) pipeline. A process of the app on the client device issues a task request by "clicking" a monadic hyperlink. The computility handles the task request and returns a response, which is consumed by the app process. During consumption, the app process may issue zero or more subtask requests to the computility.

The computility follows three principles of requirements (architectural constraints). We first highlight their main concepts and then elaborate on them in Subsections 2.1, 2.2, and 2.3, respectively.

- *Global Resource Space (GRS).* The computility has one global resource space. Any resource in the world is able to join this space unconstrained. Any task can only use resources in the GRS. The design of GRS needs to strike a balance between being compatible and unconstrained.
- *Stored-Computer Architecture (SCA).* Any task execution needs data, program, and computer resources. The first two types are already supported by the stored-program concept, providing coded information and coded algorithm capabilities. The computility needs to additionally support the coded system capability, such that a hypertask can be executed on an elastic cluster, which is a set of dynamically constructed, elastic, and full-stack resources.
- *Monadic Hypermedia (MH).* The hyperlink or hypermedia concept in the WWW is extended to a monadic hyperlink/hypermedia concept for computility, to support task composition and decoupling of systems operations from app logic code.

2.1 Global Resource Space

The GRS principle inherits the unconstrained, global "information" space concept of WWW, but extends it to a global "resource" space to accommodate all types of resources, including information resources as well as computing resources.

When designing the resource space of computility, there is a key tension between reducing constraints and maintaining compatibility when matching tasks

①https://sky.cs.berkeley.edu/, Feb. 2025.

to resources. It is worth noting that when formulating and thinking about requirements of "being unconstrained" and "being compatible", we should not take them as a categorical yes/no metric but as a spectrum of pursuits beneficial for the wellbeing of the computing utility ecosystem.

The dichotomy way of thinking has its merits and place. For instance, Fig.2 shows a clear and simple classification of computility designs into four cases: incompatible and constrained (IC), compatible and constrained (CC), incompatible and unconstrained (IU), and compatible and unconstrained (CU). Apparently, we should avoid the worst case of IC and strive for the best case of CU.

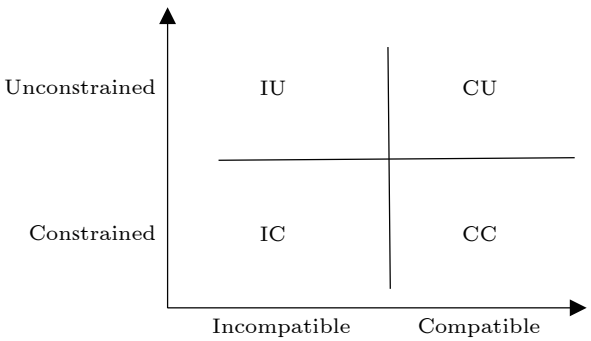

Fig.2. Four cases of constraint and compatibility.

However, Fig.2 is not a totally correct way to categorize and think about being unconstrained and being compatible. Each of them is a spectrum and a continuum. There are more than four cases (IC, CC, IU, and CU). In fact, there are potentially infinitely many cases. One computility design can be more compatible or less constrained than another.

For instance, as shown in Fig.3, one can argue that when accessing files on the Internet, the WWW

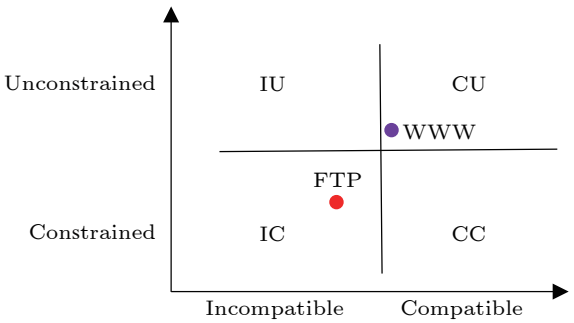

Fig.3. Perspectives of constraint and compatibility.

design is less constrained and more compatible than the FTP technology. It is not appropriate to say that FTP is incompatible and constrained, while the WWW is compatible and unconstrained.

All resource spaces in practical use have constraints, some of which we have to oblige, such as subscription walls, security walls, privacy protection, as well as sovereignty laws and regulations. What we want is to reduce artificial constraints and complexity, via better abstractions and technologies.

To reduce provider-side constraints, the GRS should allow any organization or individual to add any resource to the computility without permission from a third party, similar to how a website can add a webpage to the WWW without constraints. The organization or individual is called the resource owner.

To reduce consumer-side constraints, the GRS should accommodate diverse task-resource matching schemes and resolve compatibility issues. All schemes should try to satisfy the one-click instant access property. Four types of matching schemes are discussed below and illustrated in Fig.4.

- *By Name*. When a resource joins the computili-

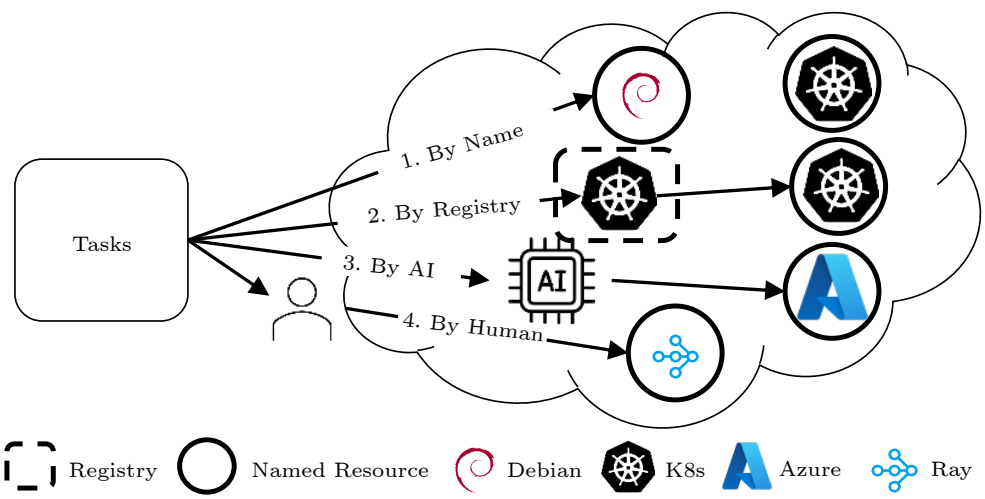

Fig.4. Four matching schemes in the global resource space.

ty, it is assigned a globally unique name or identifier. The resource owner has the rights and responsibility of name assignment, following some global standards such as a uniform resource name (URN), URI, or uniform resource locator (URL) to achieve name uniqueness. Tasks can access a resource using its name. The approach is also used by the WWW and is similar to random access memory.

• *By Registry.* When a resource joins the computility, it may be registered in a resource registry. To access a resource, tasks first look up the resource in the registry, which returns a handle for access. A resource does not need to provide a unique name to a task. A consistent registry must be maintained in the computility. The approach is similar to associative memory.

• *By AI.* Suppose there is neither global naming nor registry. One can still resolve compatibility issues assuming the existence of an oracle, which returns desired resource handles when receiving a task request. Techniques such as search engines, targeted advertising, and large language models (LLMs) can be leveraged to build a matching oracle.

• *By Human.* When all automatic tools fail, human involvement is needed for resolving computability issues. But the computility design, especially the GRS design and abstractions, should try to help the user/developer minimize human efforts required.

Learning from telephony, we also need some form of task switching mechanism to match tasks and resources in the global resource space. Early telephone switches use human operators to match and connect the two parties of a conversation. Modern telephone switches not only automatically connect two parties, but also try to provide good enough quality-of-service on an elastic virtual channel between them.

The problem of task switching is similar: match and connect the two parties of every task execution, where one party is a task with its resource specification, and the other party is the set of resources matching the specification. We can imagine the task switching problem as a dynamic bipartite graph of task nodes and resource nodes. At any time, there may be billions of tasks looking for billions of resources in the computility.

Unlike packet switching, task switching does not necessarily mean routing a task to a resource. The key point is the result of task switching: a task is started and executed on resources matching the task's resource specification. Similar to dining options, one can go to a restaurant, order a catering service, or take out.

When reducing constraints, we should not forget to resolve compatibility issues that arise in task-resource matching. Compatibility first means that we want a right match: a task can start on a resource. The probability of "wrong matching", equivalent to "wrong number" in telephony or 404 status code in the WWW, should be exceedingly low. Second, compatibility means that we want a good match, akin to the QoS objective in the telephone switch: task executions on resources should exhibit high goodput, low tail latency, and low cost. Therefore, compatibility is not a categorical metric but a continuum where one system can be more compatible than another.

2.2 Stored-Computer Architecture

Historically, computers up to the ENIAC (Electronic Numerical Integrator and Computer) followed a fixed-program architecture, shown as phase 1 in Fig.5. In such a system, information is coded and stored as data in memory, but the "program" is built into the system hardware.

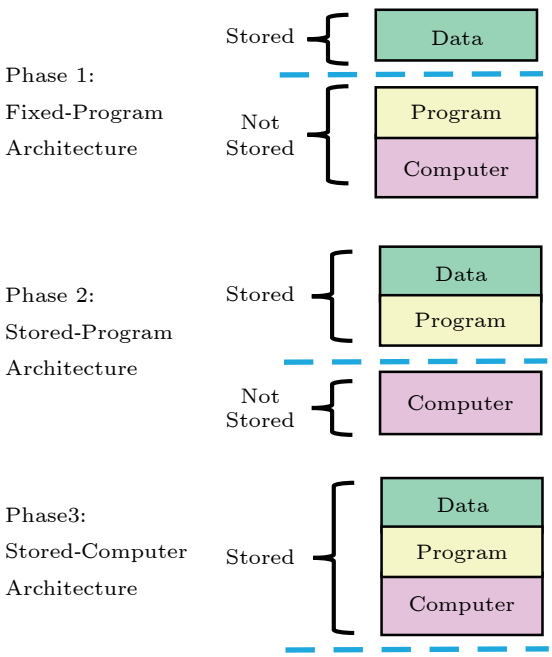

Fig.5. Three phases of computing systems evolution.

The defining concept of modern computers is stored-program architecture, shown as phase 2 in Fig.5. Algorithms are coded as programs and stored in memory as data. The stored-program concept enables general-purpose computers, which fetch and execute instructions one by one. It also facilitates manipulation of programs in the same way that the computer manipulates data, which enables program com-

pilation, operating systems, and static/dynamic libraries.

Any task execution needs data, program, and computer resources. The hypertasking model takes the position of stored-computer architecture, shown as phase 3 in Fig.5. That is, all three types of resources (data, program, and computer) are coded and stored in the computility, such that the matched resources can be dynamically provisioned to suit the needs of a task. We observe that dynamic provisioning is equivalent to automatic construction of an elastic, full-stack, virtual cluster for the task. Automatically constructing the elastic cluster means automatically scaling and constructing resources at any abstraction level. The construction code and resources are reusable by any person with programming and runtime abstraction support, including static and dynamic resource type checking.

What does "coded" mean?

When discussing the three phases of computing systems evolution in Fig.5, we use phrases such as coded information, coded algorithm, and coded system. For any instance of any of the three types of resources, "coded" means that the resource is precisely described by a computer language, which could be a programming language or a not Turing-complete language, such as a markup language. The precisely described resource, i.e., coded resource, is stored, accessed, and manipulated as data. This clarification is borrowed from Donald Knuth, who considered that programs are algorithms expressed by a computer language.

With the SCA support, when any task request is sent to the computility, an elastic, full-stack, and virtual cluster is created and dedicated to the lifecycle of the task execution. This dedicated computer can be called the elastic cluster for the task.

Four characteristics of the dynamically constructed computer are briefly discussed below.

- *Cluster*. The resources are not a set of unrelated members, but form one computer system with necessary processors, memory, storage, and I/O hardware capabilities, as well as needed software and data. This could be a sequential or parallel computer, or a distributed system. This computer can be managed as a single entity.
- *Virtual*. This is a virtual computer, comprised of abstract components (coded resources) that can be mapped to physical resources, including data, programs, and computers. Thus, the stored-computer architecture can also be called stored-resource architecture.
- *Full-Stack*. The computer is composed of a full stack of resources needed by the task. The stack could be as shallow as a bare-metal server, or as deep as a multi-layered stack consisting of resources from the bare-metal, OS, middleware, libraries, up to app framework, or even SaaS layer. Each layer could have its own abstractions. The SCA should facilitate the users to program with high-level abstractions.
- *Elastic*. During a task's lifecycle, resources needed by the task may change. Therefore, the cluster organization, both virtual and physical, may change as well, including its parallelism, interconnection, synchrony, and heterogeneity, among other factors.

2.3 Monadic Hypermedia

The concept of stored-computer architecture offers benefits for computility. However, it could significantly increase programming difficulty, if the user/developer has to provide all the system coding details, in addition to information coding and algorithm coding. We need coded abstractions to hide systems details.

We propose a principle for such coded abstractions, called monadic hypermedia, shown in Fig.6. A program running on computility consists of three types of code: app functions (e.g., parallel quick sort code and data), non-app functions (e.g., parallelism details), and non-functional code (e.g., access control and resource accounting details). Preferably, the user only needs to provide the app functions, as shown in Fig.6(a), which specifies that the task consists of a quicksort function f, which maps input data a to sorted data b. But what is executed by computility is more detailed, as shown in Fig.6(b). The system actually produces mb, where m is a monad. The monadic hypermedia subsystem of computility is expected to automatically provide the purple parts of Fig.6(b), by

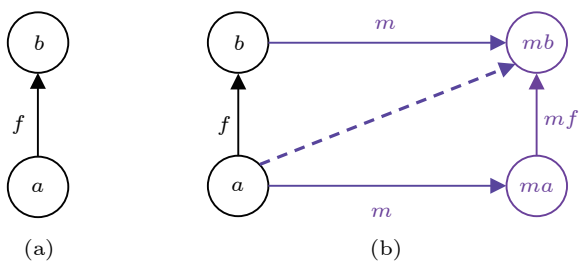

Fig.6. Illustration of monadic hypermedia via a Kleisli category. (a) App. (b) App logic and systems details

code and data completion, with minimal or no input from the user.

The monadic hypermedia works more like a decorator than a typical library call that needs to be inserted into business logic. In other words, monadic hypermedia adds systems details to app logic and realizes its composition in a non-intrusive way.

The "monadic" in monadic hypermedia tries to transfer the monad idea in category theory to utility computing[17]. The computility should provide built-in types of monadic hypermedia, to manage common systems details related to naming, accounts, access control, effects, heterogeneity, and exceptions. These call for careful abstraction and design, beyond brute force listing and implementation. Abstraction for built-in monadic hypermedia targeting lifecycle management of tasks could be a good starting point. We should also note that programming language support may not be enough. Runtime support needs to be provided to ensure the correctness and compositionality of monadic hypermedia[18]. Additionally, mechanisms should be established to allow third parties to define their own monadic hypermedia, and add them to the global resource space.

3 Experiments on a Prototype of Computility

This section provides initial supporting evidence for the hypertasking computing model, by outlining a prototype architecture and conducting Things-Multicloud experiments. The prototype is called PoC as a shorthand for the Prototype of Computility. It realizes the request-response-consume (RRC) pipeline (shown in Fig.7), via three abstractions of service assembly, grip, and named resource (shown in Fig.8).

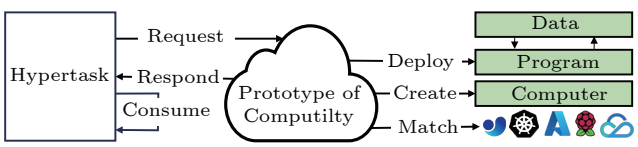

Fig.7. Execution of request-response-consume pipeline.

3.1 Resource Provisioning

The Prototype of Computility (PoC) must have resources to execute hypertasks. Any resource on the Internet can join the PoC to become part of its named resource space (NRS) and is assigned a globally unique name, such as a URI, needing no permission from a third party. However, the resource provider needs to execute the NRS server software. A resource collection, such as a cloud or an HPC (high-performance computing) center, may need to run only one NRS server. This is similar to how a Web site needs to run a Web server to enable the world to access its resources.

As shown in Fig.9, experiments are conducted on resources of four sites, using six virtual machines (VMs) from each of Azure Cloud[②], Tencent Cloud[③], and a private cloud. Each VM of the Azure Cloud or the Tencent Cloud is equipped with four CPU cores, 16 GB memory, and 100 Mbps public network bandwidth. Each VM from the private cloud has 16 CPU cores, 32 GB memory, and a public network bandwidth of 100 Mbps. The round-trip time between our Things Lab and the three clouds is as follows: 188 ms to Azure, 35 ms to Tencent, and 42 ms to the private cloud. The required resources are described in hypertasks, as shown in Fig.10 and Fig.11.

As shown in Fig.12, over a dozen types of resources on the Internet are used by the PoC and our experiments. They are named resources. Once a resource is in the NRS, it will be accessible worldwide. For instance, Fig.8 only shows how the YOLO11-based[19] image recognition application uses the named resources. However, a matrix multiplication application can also use these named resources.

3.2 Hypertask Description

A hypertask is described by a service assembly, in addition to the real app code. For instance, a matrix multiplication task is comprised of the real matrix multiplication code mm.py and the associated service assembly file mm.ht, as shown in Fig.11(d). They are sent to the PoC as a whole in one task request.

The hypertasks used in the Things-Multicloud experiments are shown in Fig.8, 10, and 11. The PoC provides four types of built-in monadic hypermedia, including scheduler, task closure, environment, and lifecycle management. This enables service assembly description to reduce programming complexity, by helping programmers focus on app functions and decoupling tasks from specific computing resources.

A service assembly description of a hypertask contains five components, as shown in Fig.10 and Fig.11. The *Business* component links to the app functions of the hypertask, which will be deployed to an execu-

[②]Azure Cloud. https://azure.microsoft.com/, Feb. 2025.
[③]Tencent Cloud. https://cloud.tencent.com/, Feb. 2025.

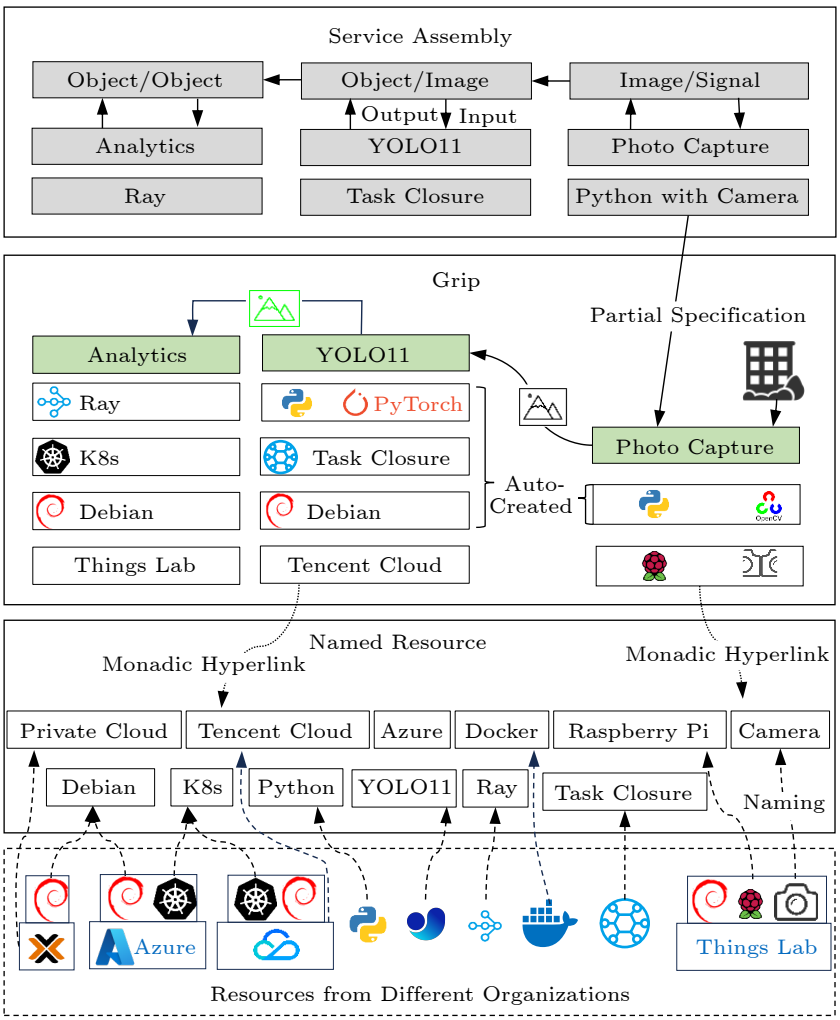

Fig.8. Architecture of the Prototype of Computility (PoC).

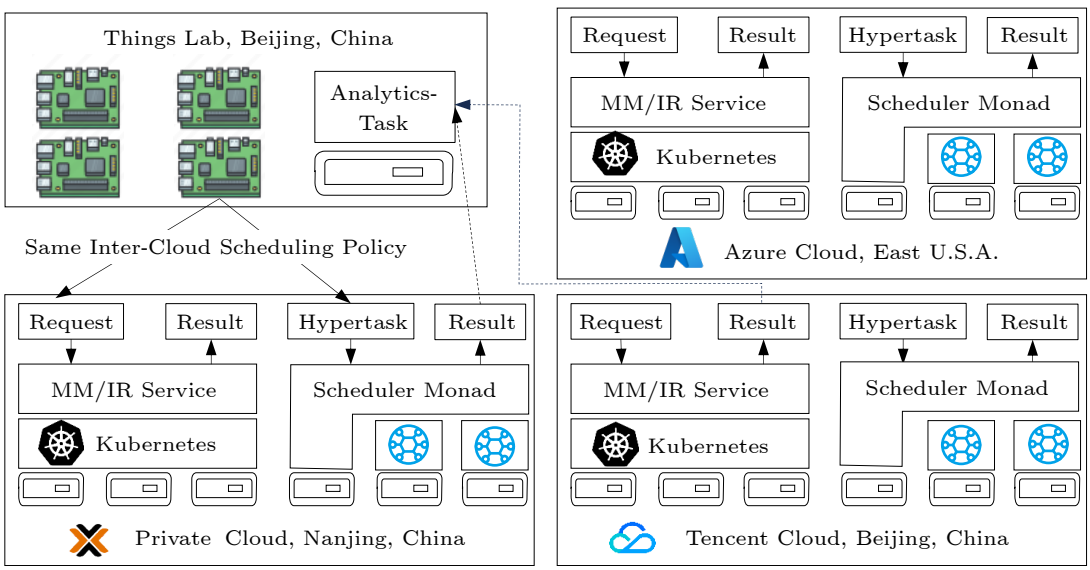

Fig.9. Experiments running on resources of four sites.

tion environment specified by *Environment*. *Input* and *Output* refer to the data required by the app functions and the data it produces, respectively. The *Assembly* component composes existing hypertasks to

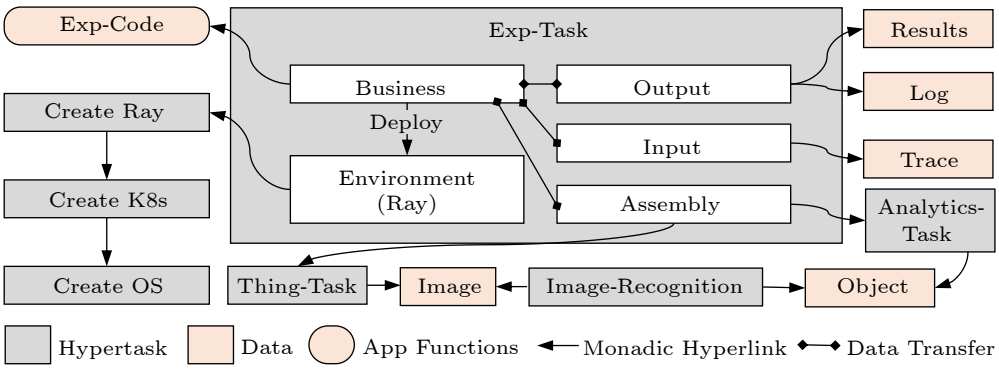

Fig.10. Structure of the experiment hypertask.

```
1 Name: Exp-Task
2 Environment:
3   type: Ray
4 Assembly:
5   thing1:
6     name: Thing-Task
7     depends_on: analytics
8     monad: scheduler-monad
9   thing2: # same as thing1
13  thing3: # same as thing1
17  thing4: # same as thing1
21  analytics:
22    name: Analytics-Task
23 Business: Exp-Code
24 Input: { "Trace":"" }
25 Output: { "Results":"", "Log":"" }
```

(a)

```
1 Name: Thing-Task
2 Environment:
3   type: OS
4   provider: things
5 Business: Photo-Capture-Code
6 Input: { "Send_Addr":"", "Trace":"" }
7 Output: {"Hypertasks":[], "Image":"" }
```

(b)

```
1 Name: Image-Recognition
2 Environment:
3   type: Task-Closure
4 Business: YOLO11-Code
5 Input: { "Image":"", "Send_Addr":"" }
6 Output: { "Object":"" }
```

(c)

```
1 Name: Matrix-Multiplication
2 Environment:
3   type: Task-Closure
4 Business: Matrix-Multiplication-Code
5 Input: { "Send_Addr":"" }
6 Output: { "Object":"" }
```

(d)

```
1 Name: Analytics-Task
2 Environment:
3   type: Ray
4 Business: Analytics-Code
5 Input: { "Object":"" }
6 Output: { "Result":"", "Log":"" }
```

(e)

Fig.11. Five hypertasks in experiments. (a) Hypertask of the entire experiment (experiment.ht). (b) Hypertask running on IoT devices (thingTask.ht). (c) Image recognition hypertask (ir.ht). (d) Matrix multiplication hypertask (mm.ht). (e) Analytics hypertask (Analytics.ht).

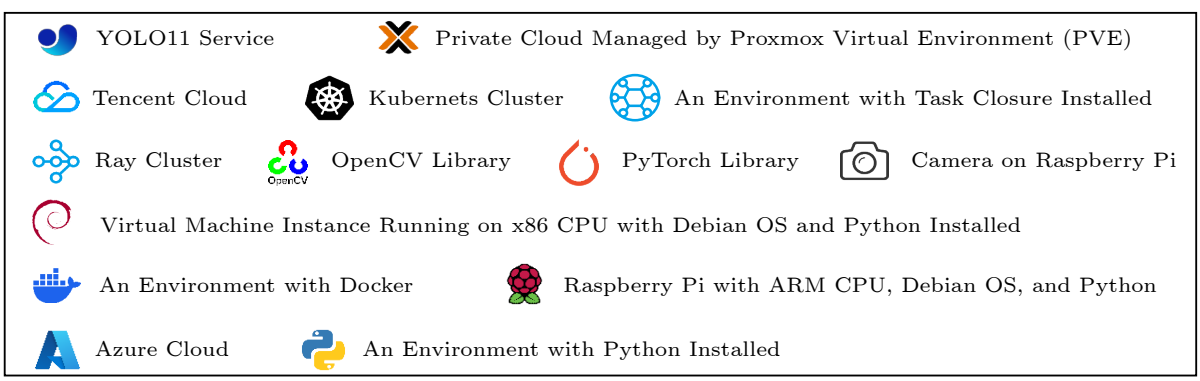

Fig.12. Named resources used in the PoC and experiments.

express a more comprehensive logic. *Environment* and *Assembly* contain non-app functions and non-functional code of the hypertask.

The Things-Multicloud experiments compare the PoC-based Things-Multicloud system with a Kubernets-based Things-Multicloud system (called the K8s[④] system). They are conducted by running the same workload consisting of an image recognition application and a matrix multiplication application. This mixed workload is chosen to see the impact of inter-application interferences. In each test, the K8s system and the PoC system use the same bare-metal resources, i.e., they have the same computing resources and network resources.

In the image recognition application, the four IoT devices in the Things Lab, each equipped with a camera, periodically capture images and send image recognition (IR) tasks to the multicloud to recognize and analyze. In the matrix multiplication application, the four IoT devices in the Things Lab periodically send matrix multiplication (MM) tasks to the multicloud to compute.

The experiments test burst load and mixed load, and calculate the task's end-to-end latency and the yield of each test. Yield[4] refers to the ratio of the number of tasks meeting the user's latency requirement to the total number of tasks. In the experiments, the latency requirement is set to 1 second. Burst load tests the system's elasticity, while mixed load tests whether the system can reduce interference between tasks.

In the burst load scenario, identical computing tasks (MM or IR) are repeatedly dispatched by all four IoT devices at specified intervals, ranging from 62.5 ms to 1 s, for 40 seconds.

In the mixed load scenario, the IoT devices submit both types of tasks simultaneously. During the first half of the test, the ratio of MM to IR is set at 1:R. Conversely, in the second half, the ratio is adjusted to R:1. The experiment assesses two different cases with R values of 4 and 8, respectively. The experiments test when the total number of tasks ranges from 3 200 to 25 600 for 200 seconds, a period sufficient for K8s to perform multiple scaling-ups.

3.3 Task-Resource Binding

From Fig.8 and Fig.11, we can see that gaps exist between high-level hypertask descriptions and low-level named resources. Mechanisms are needed to close these gaps. The core need is to create and maintain an elastic cluster to run a hypertask.

The PoC offers a grip abstraction. It automatically and recursively builds low-level resources, assembles them into a full-stack resource, and runs app functions on this resource, i.e., an elastic cluster. The PoC system currently supports the automatic construction of virtual machines, operating systems, Docker[⑤], K8s, Ray[⑥] cluster, and task closure. The last is a built-in monadic hypermedia of the PoC.

The property of maintaining an elastic cluster is validated through the mixed load scenario tests. In the first phase, the PoC allocates more resources to IR tasks than to MM tasks. When the task ratio changes, the PoC converts the resources used by IR tasks into resources for MM tasks. It obtains new resources from multiple clouds on demand when the resources needed by tasks are not sufficient.

3.4 Performance Results

It appears that the PoC system provides a layer of indirection between tasks and resources to support programmability (e.g., automatic build). We must ask 1) how large the overhead is, and 2) what the overall system performance is when executing tasks.

Table 1 shows the latencies of constructing different types of resources in the PoC following the hypertasking computing model, compared with latencies using traditional methods on existing clouds. The results indicate that the overheads are within 0.31% to 6.68%. There is room for improvement, but such overheads are acceptable.

Table 1. Latency in Constructing Resources at Different Abstraction Levels

Resource	Hypertasking (s)	Traditional (s)
Tencent Cloud VM	39.97	38.46
Azure Cloud VM	58.02	55.65
Private Cloud VM	46.94	44.43
Docker	70.30	65.90
K8s (3 nodes)	403.20	396.80
Ray cluster	57.84	57.66

Fig.13 and Fig.14 show the performance of the PoC system and the K8s system under the burst and

④Kubernetes. https://kubernetes.io/, Feb 2025.
⑤Docker. https://www.docker.com/, Feb 2025.
⑥Ray. https://docs.ray.io/, Feb 2025.

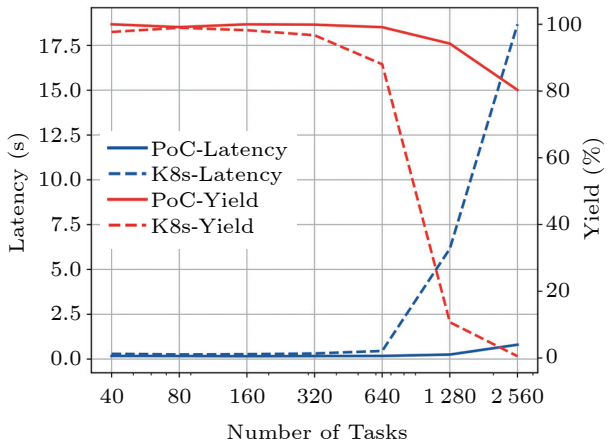

Fig.13. Application performance on the PoC vs the K8s under burst load in multicloud.

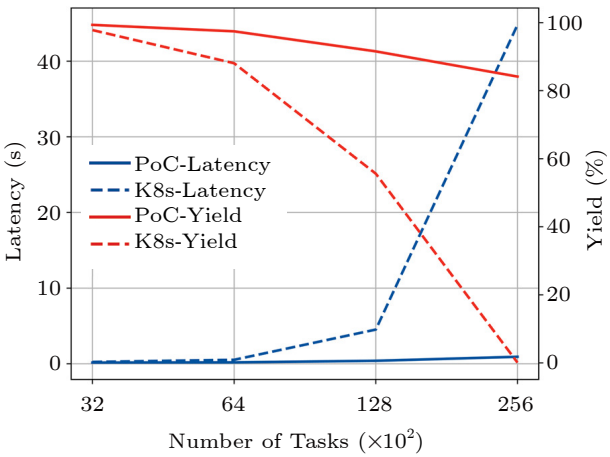

Fig.14. Application performance on the PoC vs the K8s under mixed load in multicloud.

mixed loads in the environments, respectively. The results show that the PoC system can significantly reduce application latency and improve application yields.

Under the burst load, as shown in Fig.13, the PoC system consistently demonstrates lower latencies than the K8s system, achieving an average reduction of 59% and a peak reduction of 95%. The yield has improved by an order of magnitude, 22 times on average.

In the MM test case with 320 tasks, the PoC demonstrates a slightly higher p99 tail latency compared with the K8s system. This is attributed to the queuing of tasks on the private cloud prior to the expansion of containers, which prompts the scheduler monad to redistribute the load to other clouds, thereby increasing latency. In contrast, due to HTTP timeout setting, the K8s-based application does not schedule the task to other clouds. In future work, more accurate execution time estimates will be incorporated into the scheduler monad. Users can improve application performance without modifying the application's hypertasks.

Under the mixed load, as shown in Fig.14, the PoC system reduces latencies by 72% on average and by 97% in the best case, compared with the K8s system. The PoC system also improves yield by orders of magnitude at high loads.

4 Related Work

Grid computing[6, 7] was proposed in the 1990s and designed as a computing infrastructure that provides on-demand computing resources from various organizations. Various grid computing approaches were proposed, implemented, and used, especially in the scientific computing community. They aim to connect geographically distributed resources via the Internet and integrate them into a large supercomputer, which provides computing, storage, and data resources. Multiple resource providers form a federation or a virtual organization.

Cloud computing[8, 9] is the most recent production-grade development of computing utility, which delivers computing resources such as servers, storage, software, and databases over the Internet, allowing users to access these services on demand without owning physical infrastructure. It operates on a scalable, pay-as-you-go model, making it cost-efficient and flexible for businesses and individuals. Cloud services are typically categorized into Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). They can be deployed via public, private, and hybrid models. Cloud computing does not solve problems such as platform lock-in, API explosion, and low resource utilization.

Sky computing[7, 10–13] and joint-cloud computing[20] are two approaches to the next stage of utility computing. Sky computing hopes to connect multiple cloud vendors to provide a non-differentiated commodity that forms a public utility computing service. Through a compatibility layer, sky computing hides differences in implementation between clouds and allows users to run tasks on different clouds without modification. It introduces the intercloud layer that decouples users from cloud vendors and runs jobs on the cloud (or clouds) that meet users' requirements, like price and performance. The intercloud layer is

implemented by an intercloud broker, which creates a two-sided market between users and clouds. Joint-cloud computing focuses on cloud collaboration, allowing users to customize cloud services through a "software-defined" approach. It supports the exchange of service capabilities through the service catalog of the Joint-Cloud Computing Environment.

As a testbed for future cloud computing systems, Cloudlab[21] supports researchers in building customized clouds through a Geni profile, which accurately describes hardware resources, network topology, and building scripts. Cloudlab provides three types of resources, including virtual machines, bare-metal machines, and containers. FABRIC[22] is a national network research infrastructure that provides the everywhere-programmable capability and supports access to supercomputing and cloud platforms. Infrastructure as Code (IaC) offers benefits for infrastructure automation by enabling automated, efficient, and consistent environment creation through declarative configurations. Terraform[23], as a popular IaC tool, provisions and manages resources across different clouds with the requirement of using provider-specific APIs.

Computing power network[24] is an emerging concept of connecting distributed and heterogeneous computing nodes. It focuses more on networking technology, including resource connecting, scheduling, and compute first networking[25]. Our work aims to build a global computing system that satisfies the six salient features, including pay-per-use services, planet-scale culture, low-entropy systems, abstraction-powered programs, high-goodput utility, and one-click instant access.

5 Conclusions

John McCarthy and Barbara Liskov proposed the vision of computing utility and Internet Computer, respectively. To realize the McCarthy-Liskov vision, this paper outlines a hypertasking computing model to extend the World Wide Web to the World Wide Computing Utility. The hypertasking model lowers the entry barrier for users to utilize worldwide information and computing resources by having them send hypertasks to the computing utility through a request-response-consume pipeline and adhering to three architectural constraints. The global resource space constraint allows unconstrained resource joining and accessing while supporting compatible task-resource mapping. The stored-computer architecture constraint provides the ability to dynamically construct and maintain a virtual, full-stack, and elastic cluster dedicated to each hypertask. The monadic hypermedia constraint provides abstractions to reduce programming complexity.

We built a prototype to validate the hypertasking model within a Things-Multicloud scenario, using resources from four sites across China and the United States. During experiments, the prototype dynamically constructed a full-stack elastic cluster and deployed tasks on the cluster, such as automatically building a Ray cluster from bare-metal and adjusting the CPU occupation from 24 cores to 72 cores according to the task demand. Initial evidence from experiments suggests that the prototype introduces acceptable overhead for resource auto-building, ranging from 0.31% to 6.68%. Compared with Kubernetes, the prototype shows a higher throughput for tasks that meet users' latency requirements.

In future work, new naming and access mechanisms for global resources will be developed, along with programming and runtime support for stored-computer architecture. A variety of monadic hypermedia will be created to suit common scenarios. Additionally, larger-scale experiments are planned to thoroughly verify the hypertasking model, including testing applications across multiple domains, such as education and large AI models.

Conflict of Interest Zhi-Wei Xu is the Editor-in-Chief for Journal of Computer Science and Technology and was not involved in the editorial review of this article. The authors declare that there are no other competing interests.

References

[1] McCarthy J. Time sharing computer systems. In *Management and the Computer of the Future*, Greenberger M (ed.), MIT Press, 1962, pp.221–236.

[2] Liskov B. The power of abstraction. In *ACM Turing Award Lectures*, Association for Computing Machinery, 2011, p.2008. DOI: 10.1145/1283920.1962421.

[3] Xu Z, Li G, Sun N. Information superbahn: Towards new type of cyberinfrastructure. *Bulletin of Chinese Academy of Sciences*, 2022, 37(1): 46–52. DOI: 10.16418/j.issn.1000-3045.20211117008. (in Chinese)

[4] Xu ZW, Li ZY, Yu ZS, Li FZ. Information superbahn: Towards a planet-scale, low-entropy and high-goodput

computing utility. *Journal of Computer Science and Technology*, 2023, 38(1): 103–114. DOI: 10.1007/s11390-022-2898-7.

[5] Berners-Lee T. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper Business, 2000.

[6] Foster I, Kesselman C. The Grid 2: Blueprint for a New Computing Infrastructure (2nd edition). Morgan Kaufmann, 2003. https://dl.acm.org/doi/book/10.5555/996313.

[7] Keahey K, Tsugawa M, Matsunaga A, Fortes J. Sky computing. *IEEE Internet Computing*, 2009, 13(5): 43–51. DOI: 10.1109/MIC.2009.94.

[8] Comer D. The Cloud Computing Book: The Future of Computing Explained. Chapman and Hall/CRC, 2021.

[9] Jonas E, Schleier-Smith J, Sreekanti V, Tsai C C, Khandelwal A, Pu Q, Shankar V, Carreira J, Krauth K, Yadwadkar N, Gonzalez J E, Popa R A, Stoica I, Patterson D A. Cloud programming simplified: A Berkeley view on serverless computing. arXiv: 1902.03383, 2019. https://arxiv.org/abs/1902.03383, Feb. 2025.

[10] Fortes J A B. Sky computing: When multiple clouds become one. In *Proc. the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, p.4. DOI: 10.1109/CCGRID.2010.136.

[11] Stoica I, Shenker S. From cloud computing to sky computing. In *Proc. the 2021 Workshop on Hot Topics in Operating Systems*, Jun. 2021, pp.26–32. DOI: 10.1145/3458336.3465301.

[12] Yang Z, Wu Z, Luo M, Chiang W L, Bhardwaj R, Kwon W, Zhuang S, Luan F S, Mittal G, Shenker S, Stoica I. SkyPilot: An intercloud broker for sky computing. In *Proc. the 20th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2023, pp.437–455.

[13] Stoica I. Sky computing: Opportunities and challenges. In *Decision Making and Decision Support in the Information Era*, Balas V E, Dzemyda G, Belciug S, Kacprzyk J (eds.), Springer, 2024, pp.15–27. DOI: 10.1007/978-3-031-62158-1_2.

[14] Xu Z, Chao L, Peng X. T-REST: An open-enabled architectural style for the Internet of Things. *IEEE Internet of Things Journal*, 2019, 6(3): 4019–4034. DOI: 10.1109/JIOT.2018.2875912.

[15] Chao L, Peng X, Xu Z, Zhang L. Ecosystem of things: Hardware, software, and architecture. *Proceedings of the IEEE*, 2019, 107(8): 1563–1583. DOI: 10.1109/JPROC.2019.2925526.

[16] Sun NH, Zhang YQ, Zhang FB. How to translate 算力 (Suanli) into English? *Communications of the CCF*, 2022, 18(9): 87. (in Chinese)

[17] Milewski B. Category Theory for Programmers. Blurb, 2019.

[18] Ma H, Qiao Y, Liu S, Yu S, Ni Y, Lu Q, Wu J, Zhang Y, Kim M, Xu H. DRust: Language-guided distributed shared memory with fine granularity, full transparency, and ultra efficiency. In *Proc. the 18th USENIX Symposium on Operating Systems Design and Implementation*, Jul. 2024, pp.97–115.

[19] Khanam R, Hussain M. YOLOv11: An overview of the key architectural enhancements. arXiv: 2410.17725, 2024. https://arxiv.org/abs/2410.17725, Feb. 2025.

[20] Wang H, Shi P, Zhang Y. JointCloud: A cross-cloud cooperation architecture for integrated Internet service customization. In *Proc. the 37th IEEE International Conference on Distributed Computing Systems*, Jun. 2017, pp.1846–1855. DOI: 10.1109/ICDCS.2017.237.

[21] Duplyakin D, Ricci R, Maricq A, Wong G, Duerig J, Eide E, Stoller L, Hibler M, Johnson D, Webb K, Akella A, Wang K, Ricart G, Landweber L, Elliott C, Zink M, Cecchet E, Kar S, Mishra P. The design and operation of CloudLab. In *Proc. the 2019 USENIX Annual Technical Conference*, Jul. 2019, pp.1–14.

[22] Baldin I, Nikolich A, Griffioen J, Monga I I S, Wang K C, Lehman T, Ruth P. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, 2019, 23(6): 38–47. DOI: 10.1109/MIC.2019.2958545.

[23] Brikman Y. Terraform: Up and Running: Writing Infrastructure as Code (3rd edition). O'Reilly Media, 2022.

[24] ITU-T. Computing power network—Framework and architecture. ITU-T-Y. 2501. International Telecommunication Union, 2021. https://standards.globalspec.com/std/14474111/y-2501, Feb. 2025.

[25] Gong X, Bai C, Ren S, Wang J, Wang C. A survey of compute first networking. In *Proc. the 23rd IEEE International Conference on Communication Technology*, Oct. 2023, pp.688–695. DOI: 10.1109/ICCT59356.2023.10419572.

Zhi-Wei Xu received his Ph.D. degree from the University of Southern California, Los Angeles. He is a professor of the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research areas include high-performance computer architecture and distributed systems.

Zi-Shu Yu is a Ph.D. candidate of Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He received his B.E. degree in computer science and technology from University of Chinese Academy of Sciences, Beijing, in 2019. His current research interests include distributed systems and runtime management.

Feng-Zhi Li is a Ph.D. candidate of Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He received his B.E. degree in computer science and technology from University of Chinese Academy of Sciences, Beijing, in 2020. His current research interests include distributed systems and application packaging.

Yao Zhang is a Ph.D. candidate of Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He received his B.E. degree in computer science and technology from Peking University, Beijing, in 2022. His current research interests include distributed systems and resource management.