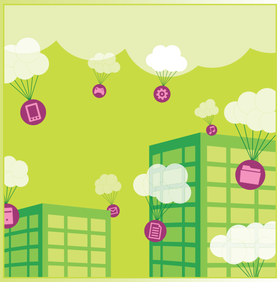


Challenges and Software Architecture for Fog Computing

This article presents a detailed description of fog computing (also known as edge computing) and explores its research challenges and problems. Based on the authors' understanding of these challenges and problems, they propose a flexible software architecture, which can incorporate different design choices and user-specified polices. They present their design of WM-FOG, a computing framework for fog environments that embraces this software architecture, and evaluate their prototype system.

Zijiang Hao
College of William and Mary

Ed Novak
Franklin and Marshall College

Shanhe Yi and Qun Li
College of William and Mary

The explosive proliferation of ubiquitously connected devices has revolutionized every aspect of human life. However, because most end-user devices, such as smartphones, tablets, and wearable devices, have generally weaker CPUs, limited network connectivity, less memory and storage, and so on, applications nowadays are usually backed by cloud services. Cloud computing has significantly changed the way we leverage resources for computation, networking, and storage. It provides resources on an on-demand basis, saves expenditures on hardware, and breaks down various technical barriers. Resource pooling in data centers indeed provides more than enough resources for end-user devices. However, moving data from the edge of Internet to the core of Internet, where most data centers are located, isn't easy, especially when more and more data is user-generated content that requires high bandwidth to transmit.[1] In addition, unpredictable delays might ruin the user experience of delay-sensitive applications, such as human-computer interfaces, emergency services, and real-time games. While we believe that cloud computing will still be a mainstream computing paradigm in the future, the rapid development of Internet and pervasive mobile devices has called for a new computing paradigm that can overcome the inherent drawbacks of cloud computing, such as unpredictable latency, bandwidth bottlenecks, lack of mobility support and location awareness, and so on. To this end, fog computing (also known as edge computing) has been proposed as a nontrivial extension of cloud computing. By providing elastic resources at the edge of network, fog computing can better support a variety of emerging applications.

Fog computing has been defined from several perspectives;[2,3] and similar concepts such as cloudlets,[4] mobile-edge computing[5] and mobile-cloud computing[6] have also been proposed. In our previous work,[7] we defined fog computing as

> *a geographically distributed computing architecture, with a resource pool that consists of one or more ubiquitously connected heterogeneous devices (including edge devices) at the edge of network, and not exclusively seamlessly backed by cloud services, to collaboratively provide elastic computation, storage and communication (and many other new services and tasks) in virtualization isolated environments to a large scale of clients in proximity.*

Fog computing will benefit several relevant domains, including mobile/wearable computing,[8] Internet of Things (IoT), and big data analytics, in reducing latency, increasing throughput, consolidating resources, saving energy, and enhancing security and privacy.[9] In IoT, fog computing can provide unified interfaces and flexible resources to accomplish heterogeneous computational and storage requests. In virtual reality, a 3D VR gaming headset has to be cable-connected to a high-end server for low latency on processing complex 3D graphics. Fog computing can fulfill the low-latency need for VR users in proximity, and can save expenditures on extra hardware. In big data analytics, huge volumes of data are generated at the edge of network. Fog computing supports edge analytics, which can reduce the delay of data analytics and decrease the cost of data transmission and storage.

Here, we introduce fog computing, a new computing paradigm that extends cloud computing. Fog computing promises performance benefits such as low latency and quick response time in various application scenarios. We compare fog computing and cloud computing in detail, and list a number of research challenges and problems. Based on our understanding of these challenges and problems, we propose a flexible software architecture, which can incorporate different design choices and user-specified polices. We highlight WM-FOG (WM standards for the College of William and Mary), a computing framework for fog environments that embraces this software architecture. We also conduct experiments on our prototype system to show that WM-FOG can work effectively and efficiently in real-world fog environments.

Comparisons between Fog and Cloud

There are several key differences between fog and cloud computing. Cloud servers are usually rack-mounted, high-end servers located in large, warehouse-like data centers. Centralized cloud servers allow for replication, load balancing, failure recovery, power management, and easy access to failed hardware for repairing and replacement. For this reason, the reliability of cloud services can be held at a high standard. The exact opposite can be expected in fog computing. Fog nodes are geographically distributed, scattered all over the edges of Internet, and logically decentralized in that they are maintained by different organizations. Consequently, fog nodes aren't as reliable as cloud servers, and physically locating a failed fog node and repairing it is more difficult and costly. Many financial and time costs, such as those related to power and system configuration, can't be amortized as they would be with cloud computing. Another key insight is that the network connectivity to fog nodes cannot be guaranteed. An unreachable fog node can't fulfill any request even if its computational hardware is fully functional. Simple tasks such as regular testing and auditing of hardware are immensely more complicated and costly in fog computing, due to the necessary coordination among different organizations, geographical distribution of hardware, and unreliable network connectivity.

Scheduling tasks in fog computing is complex compared with cloud computing. A fog computing application is typically spread over the client's mobile device, one of potentially many fog nodes, and occasionally a back-end cloud server. Therefore, deciding where to schedule computational tasks in fog computing is more difficult. For cloud computing applications, the latency is usually predictable. For fog computing applications, however, deciding which fog node to use alters the latency that the user will experience. Besides the unpredictable round-trip time, slow hardware and low bandwidth also affect the user-perceived latency. Meanwhile, some tasks, such as aggregate caching, might benefit from running in the back-end cloud. Another problem is that it's unclear where the scheduling should occur. Entrusting the client devices to perform the scheduling opens the possibility of malicious users abusing the system. Fog nodes might act selfishly or might not always be aware of tasks running on the client devices. The back-end cloud might introduce unnecessary latency to the scheduling program. In short, in fog computing, more factors must be considered in deciding where and when to schedule tasks to provide the best user experience.

Fog nodes are maintained independently by many different organizations, which is in sharp contrast to the back-end cloud owned and maintained by a single organization. This means that fog nodes can't be trusted as easily as cloud servers. Users can more easily trust cloud computing because the organizations that provide cloud services are well-motivated to invest in resilient security and privacy measures. Fog computing, on the other hand, is implemented by many independent agents. These various owners might not maintain the same rigorous privacy and security standards, let alone the high standard fulfilled by cloud computing. As such, users will have a much more difficult time trusting different fog nodes.

Fog nodes are heterogeneous, where there's no guarantee that the nodes will contain similar resources. In fact, quite the opposite can be expected; fog nodes, owned and maintained by different organizations, usually have vastly different RAM capacity, CPU performance, storage space, and network bandwidth. This is in sharp contrast to cloud computing, in which it's common for one organization to own all of the cloud servers. To ease the burden of application deployment, hardware management and resource sharing, cloud servers usually exhibit much less heterogeneity. Furthermore, fog nodes are smaller and less powerful than large cloud servers. While a cloud server might be one of many high-end, powerful, rack-mounted servers, fog nodes are usually deployed in small batches using desktop-class machines, repurposed computing appliances such as routers and gaming consoles, and small collections of rack-mounted servers. Because of the heterogeneity and generally weaker hardware in fog computing, many of the differences we have outlined so far are exacerbated.

Fog computing aims to establish a new tier of mobile computing, in which constraints on energy and hardware resources can be relaxed by nearby fog nodes. Users can effortlessly offload computation to nearby fog nodes, and can transparently and seamlessly move computation from one fog node to another. Mobility is a key feature of the fog computing paradigm, and applications deployed on fog infrastructure need to always take this into account. This differs from cloud computing, in which applications are deployed in only one cloud at a time, unless the need for scaling is beyond the capacity of a single cloud provider. A situation that's rarely seen in cloud computing but might be common in fog computing is that users might connect to the network only briefly while moving. Consider the scenario in which fog nodes are placed along roadways and users temporarily connect to them to acquire traffic and weather conditions ahead. Another example is that users move from one fog node to another when they're leaving their office and heading to a different building on a college or corporate campus for a meeting.

Research Challenges and Problems in Fog Computing

Now that we've compared fog and cloud computing, next we discuss the challenges and problems in fog computing. The related work on the concept of fog computing includes our own efforts[10,11,7,12] and other researchers' work.[13-15] We refer the reader to these references for an integral view of the state-of-the-art on fog computing.

Fog computing is a novel computing paradigm, which demands a new programming model. We need to design intuitive and effective tools and frameworks for developers, helping them orchestrate dynamic, hierarchical, and heterogeneous resources to build compatible applications on diverse platforms. Taking task scheduling and migration as an example, various research questions might arise. How can we provide a simple abstraction for developers to mark tasks that can be migrated? What choices and preferences should be left to users? How can we allow developers to specify migration rules on various devices? Furthermore, we should avoid forcing developers to reimplement functionalities that will likely be common, such as distributed caching, workload balancing, system monitoring, and so on.

Fog computing introduces a variety of new and interesting scheduling challenges. Because tasks can now be moved between different physical devices (that is, client devices, fog nodes, and back-end cloud servers), scheduling is much more complex. Some of the research questions are as follows:

- For fog nodes with heterogeneous hardware, is it acceptable to trade energy for reduced latency?
- Should a process running on a fog node be interrupted when the user moves toward another fog node?
- How should tasks be scheduled considering latency, energy consumption, mobility, and existing workload?
- Where should the scheduling program be executed?
- What are the benefits of jointly scheduling tasks?

Beside these research questions, some other concerns must also be taken into account. For example, security and privacy considerations are complex in fog computing, and tasks from sensitive applications should be scheduled on more trustworthy nodes. Furthermore, on traditional desktop and server machines, Completely Fair Scheduling dominates the landscape. For fog computing, however, this algorithm might not be ideal, as different fog nodes could have different hardware resources, and some tasks (for example, those making the users wait) might be more important than others (for example, background services such as the backup/snapshot functionality). What other scheduling algorithms can be used to optimize the factors that are important for highly mobile computing, such as low latency?

Data management for fog computing applications also introduces new challenges. Perhaps the ideal abstraction for both users and developers is a global storage, which can always be accessed, has an infinite size, and yet performs with the speed of information stored locally. With fog computing, this dream might finally be realized. However, how to implement such a storage system is still an open question. What efficient algorithms can be used to shuffle data among devices? How can prefetching be best implemented to achieve the lowest latency? What namespace scheme should be used? How can sensitive and encrypted data be cached privately and effectively? Furthermore, energy consumption and network usage must be conserved on mobile devices, as they typically have energy limits enforced by limited battery technology and data limits enforced by mobile carriers.

An attractive aspect of cloud computing is the automatic discovery of services. As fog nodes differ from location to location, users can arrive at a new location and take advantage of the various services provided by the fog nodes in that particular location. Given that this feature relies on the prudent deployment accomplished by service developers, implementing service discovery protocols in fog computing can be quite challenging. Moreover, service provisioning is usually done dynamically in fog computing; that is, new virtual machines (VMs) are orchestrated on the spot when a particular service is needed. This raises many research questions:

- When should services be started and stopped?
- What's the best way to balance workload?
- Should VM-based or container-based virtualization be used? It might even be possible to predict what services will be needed and provision them in advance, before users even arrive.
- What methodologies should be used to efficiently provision services for hundreds, thousands, and millions of users?
- What's the best way to split workload for services that aggregate information from nearby client devices with regard to the energy efficiency on the client side?

In cloud computing, data consistency can be achieved by coordinating the cloud servers in the data centers where the cloud is deployed. In fog computing, however, things become complicated. When writing data objects in a fog environment, it's necessary to not only coordinate the back-end cloud servers, but also to invalidate the cached data on the fog nodes as well as on the client devices if strong data consistency is needed. This might result in deteriorated write performance, which weakens the benefits of using fog nodes as the write cache servers. On the other hand, fog computing also provides opportunities of achieving data consistency more efficiently than cloud computing. For example, if the write requests on a data object are sent to only one fog node during a certain time period, which we envision is a common case in fog computing, the system might temporarily transfer the data object's ownership from the cloud to the fog node. By doing this, data consistency can be achieved on the fog node, which promises better write performance than cloud computing, as the fog node resides at the edge of network. Nevertheless, fully exploiting

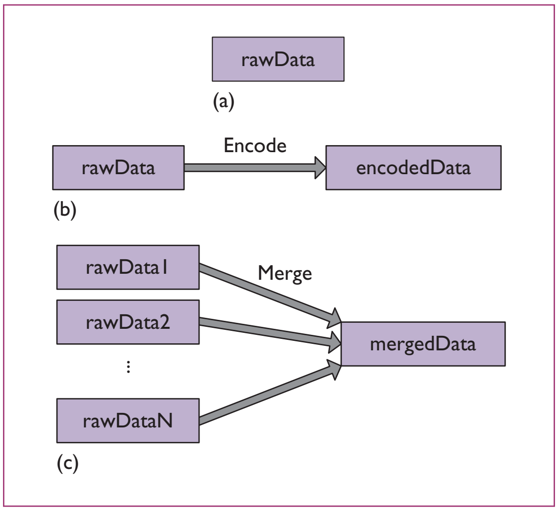


Figure 1. Workflow examples. (a) The RawVideo workflow. (b) The EncodedVideo workflow. (c) The TemperatureDistribution workflow.

the opportunities of achieving data consistency in fog computing is still challenging and requires plenty of research efforts.

Finally, and perhaps most importantly, fog computing and IoT have significant privacy and security concerns. Due to the heterogeneous nature of both, security and privacy are usually cast aside to achieve general functionality and interoperability. In other words, encryption and strict privacy policies make it more difficult for arbitrary devices to exchange data. Therefore, many manufacturers today simply do away with these features. Moreover, encryption algorithms and security protocols, which are notoriously complex, are often implemented or configured with mistakes, leaving sensitive user data exposed to attackers. This problem is further exacerbated by the dispersed ownership of fog nodes. Fog nodes are usually owned by different parties, such as universities, corporations, commonwealth organizations, and personal households. Some fog nodes might even be jointly owned by two or more parties. Users approaching a fog node might be weary of the services provided by these parties, due to their vastly differing motivations. As we've seen with online social networking, collection and resale of private user data is highly valuable to corporations. Meanwhile, these parties also need authentication protocols to protect themselves against Sybil accounts, distributed denial-of-service attacks, and other malicious activities. It will be important in the future to make fog computing applications preserve user privacy, provide rigorous security guarantees, and address the needs of all the parties involved.

Exploring solutions to the aforementioned problems is critical in realizing the many benefits promised by fog computing. It's our goal in this work to expose these design choices in detail, so that developers can more easily figure them out in their implementations. This is a first step toward identifying reasonable solutions to these problems.

WM-FOG Overview

Based on our understanding of the aforementioned challenges and problems, we propose WM-FOG, a computing framework for fog environments. The design of WM-FOG embraces a flexible software architecture, which can incorporate different design choices and user-specified polices. More specifically, WM-FOG provides a flexible way to define workflows that can be easily deployed and executed on fog-based systems. By properly scheduling the workflows on the system entities (that is, client devices, fog nodes, and back-end cloud servers), WM-FOG can take advantage of the fog computing paradigm and achieve considerable performance enhancement. Furthermore, and most importantly, WM-FOG provides a way to customize policies on the workflows, through which developers can help the system make even better use of the underlying hardware resources.

Workflow Examples

To define a workflow, the developer needs to specify its data and computation. We call the data *data items*, and the computation *transitions* in WM-FOG. Each workflow contains one or more data items and zero or more transitions.

Figure 1a illustrates a simple workflow, which is called RawVideo. This workflow contains only one data item, depicted as rawData in the figure, and no transition. The only data item, rawData, represents the raw video data that the WM-FOG system has received from a client device.

Figure 1b illustrates a slightly more complicated workflow, which is called EncodedVideo. This workflow contains two data items, rawData and encodedData, and one transition, encode. The encode transition takes rawData as input and generates encodedData as output. Clearly, in this workflow, the WM-FOG system receives

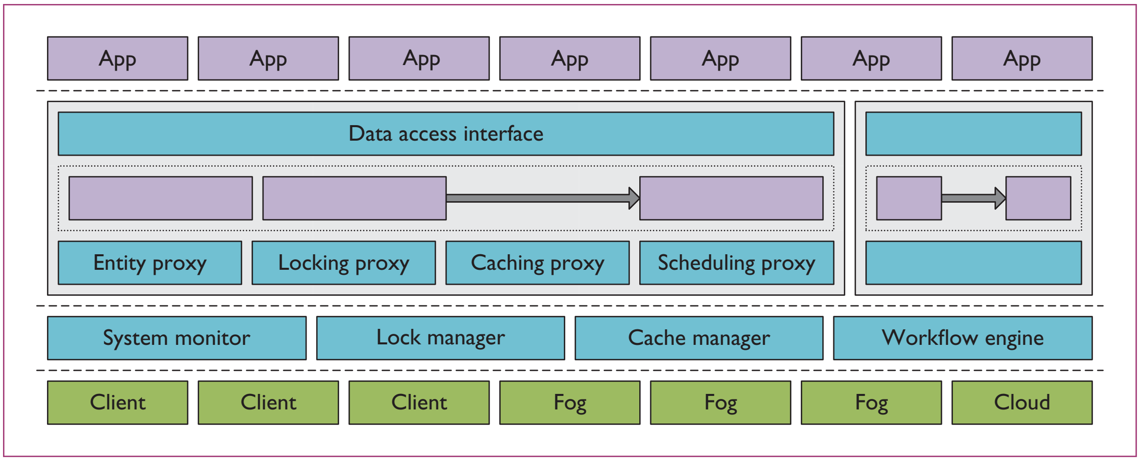


Figure 2. WM-FOG software stack. The top layer is the application layer, where user applications reside. The next layer is the workflow layer, where workflow instances reside. Under the workflow layer is the system layer, where the system components reside. The bottom layer is the entity layer, where the system entities (client devices, fog nodes, and the cloud) reside.

raw video data from a client device, encodes it, and stores the encoded data for future use.

Figure 1c illustrates a workflow involving multiple writers, which is called Temperature-Distribution. Suppose there are *N* fog nodes covering different regions, and each fog node has a number of temperature sensors deployed in the region that it covers. Each temperature sensor continuously uploads the ambient temperature data to the fog node it belongs to, and the fog node in turn forwards the data to the back-end cloud. The cloud receives data from the *N* fog nodes, merges them, and stores the merged data for future use.

System Architecture

Figure 2 depicts the architecture of WM-FOG. There are four layers in the figure. The top layer is the application layer, where user applications reside. User applications initiate workflow instances by writing input data to them, and receive results by reading output data from them. The next layer is the workflow layer, where workflow instances reside. Each workflow instance exposes a data access interface to user applications, through which its data items can be accessed. Moreover, each workflow instance has four proxies, that is, the entity proxy, locking proxy, caching proxy, and scheduling proxy. These proxies can be used to implement user-specified policies on workflows. Under the workflow layer is the system layer, where the system components — that is, the system monitor, lock manager, cache manager, and workflow engine — reside. These system components implement the fundamental mechanisms of WM-FOG, and workflow instances can communicate with them through the proxies to apply user-specified policies. The bottom layer is the entity layer, as the system entities (client devices, fog nodes, and the cloud) reside in this layer.

Customizing Workflow Policies

WM-FOG provides a workflow-defining language for developers. More specifically, developers can specify the data items and transitions for each workflow they're defining through this language. Furthermore, they can selectively implement the callback functions of the data items and transitions to define their own policies. To define a policy on a workflow, the developer is supposed to invoke the workflow's proxies in the callback functions, informing the system components of her suggestions on how to handle the workflow under various conditions. Note that the developer can only provide her suggestions, but not control the system components' behavior.

Implementing synchronization policies. A developer can implement her own synchronization policy on a data item by programming its callback functions. In these callback functions, the developer is supposed to invoke the workflow's caching proxy to communicate with the cache manager, providing her suggestions on how to synchronize the data item.

For example, suppose the developer who has defined the RawVideo workflow shown in Figure 1a has implemented a synchronization policy on the rawData data item, which eagerly synchronizes the first 20 Mbytes of data to the back-end cloud, while keeping the remaining part of data on the fog node and synchronizes it lazily. By doing this, user applications that read the data item can get served immediately. Meanwhile, the caching manager will spontaneously synchronize the remaining part of the data when the system has spare hardware resources, guaranteeing that the data item can be fully synchronized as soon as possible. This synchronization policy has the same effect on serving read requests as the default synchronization policy, which eagerly synchronizes the whole data item to the back-end cloud, but imposes less burden on the system, which is critical when the system is handling a burst of workflow requests.

Implementing locking policies. A developer can implement her own locking policy on a data item by programming its callback functions. In these callback functions, the developer is supposed to invoke the locking proxy of the workflow to communicate with the lock manager, providing her suggestions on how to manage the locks on the data item.

For example, the rawData$I$ data items (I = 1, 2, ..., N) shown in Figure 1c are written by multiple writers (that is, temperature sensors). Suppose the rawData1 data item can be written by only one writer at any time, which requires a locking mechanism to coordinate the write operations upon it. A simple way of implementing such a locking mechanism is to maintain a write lock for the data item in the back-end cloud. Before a writer writes the data item, it has to acquire the write lock from the cloud, while after the data item has been written, the writer needs to return the write lock to the cloud. Using the cloud as the centralized lock server is necessary when different writers try to acquire the same write lock from different fog nodes. However, as previously described, the rawData1 data item will only be written by temperature sensors belonging to the same fog node. In such a case, using the cloud as the centralized lock server will impose unnecessary overhead on the write operations. The developer can invoke the lock proxy in the data item's callback functions, informing the lock manager that she suggests the write lock be maintained on the fog node rather than in the cloud. By doing this, the lock manager will try to maintain the write lock on the fog node, which can improve the write performance on the data item in most cases. Note again, however, that the developer can't really control the lock manager's behavior — that is, if the lock manager considers that the write lock should be maintained in the cloud, it will do so, rather than unconditionally following the developer's suggestion.

Implementing migration policies. When defining a transition, the developer needs to specify its input data items as well as their *trigger thresholds*. For example, the encode transition shown in Figure 1b has only one input data item, rawData. Suppose the developer has specified that the trigger threshold of the rawData data item is 1,024 Kbytes when defining the encode transition. In such a case, the workflow engine will automatically invoke the onTrigger() callback function of the encode transition whenever the rawData data item has enqueued 1,024 Kbytes of data. The onTrigger() callback function is the place where the developer implements the transition's main logic.

The execution of the onTrigger() callback function is atomic in WM-FOG. In other words, the workflow engine might migrate a transition between two consecutive executions of the onTrigger() callback function, but will never do so during its execution. Data that needs to be transferred in a migration should be defined as member variables of the transition. The developer should also provide the getter and setter functions for these member variables.

A developer can implement her own migration policy on a transition, by programming its callback functions excluding onTrigger(). In these callback functions, the developer is supposed to invoke the scheduling proxy of the workflow to communicate with the workflow engine, providing her suggestions on when and how to migrate the transition.

For example, the encode transition shown in Figure 1b should be triggered mainly on fog nodes. This is because the WM-FOG system can leverage the computational power of fog nodes to achieve better performance, given that the encode transition has a good compression ratio. Nevertheless, it might be unreasonable to always trigger the transition on fog nodes, especially when they're fully loaded. For this reason, the developer might implement a migration policy on the encode transition, informing the workflow engine that she suggests migrating the transition to the back-end cloud if the fog node cannot execute it in 5 seconds. By doing this, some workload on fully loaded fog nodes can be offloaded to the back-end cloud, and the overall system performance can thus be improved.

Evaluation

Here, we give some preliminary results to show that WM-FOG can leverage the fog computing paradigm to enhance the system performance when handling workflow tasks (that is, workflow instances).

Testbed Setup

We build a testbed for our experiments. The testbed consists of five servers, one of which is more powerful than the others. We use the more powerful server as the back-end cloud server, while using the others as fog nodes. The cloud server has an 8-core Intel i7 CPU with a clock speed of 4.00 GHz and 16-Gbyte main memory. Each fog node has a 4-core CPU with a clock speed of 2.83 GHz and 4-Gbyte main memory, and directly connects to the cloud server through a 1,000 megabits per second (Mbps) network link. To simulate a real-world fog environment, we set the upper bound of the network bandwidth between each fog node and the cloud server to 40 Mbps, and the latency to 10 ms (that is, the round-trip time is 20 ms), according to the results reported by Shanhe Yi and colleagues.[7] Then we deploy our first-step implementation of WM-FOG on this testbed.

Benefits of Using Fog

We first evaluate to what extent fog computing can help when handling WM-FOG workflow tasks. To this end, we simulate a scenario in which the RawVideo workflow tasks shown in Figure 1a are handled by our system. Each RawVideo task has a total data size of 200 Mbytes, and we send it from the client device to the fog node at a transmission rate of 8 Mbps. On each fog node, the arrival intervals of RawVideo tasks follow a normal distribution with a mean of 10 seconds and a variance of 4 sec^2. We cache the 200-Mbyte rawData of each RawVideo task on the fog node, while synchronizing only the first n Mbytes of data to the cloud. The value of n varies from 0 to 200 in our experiments.

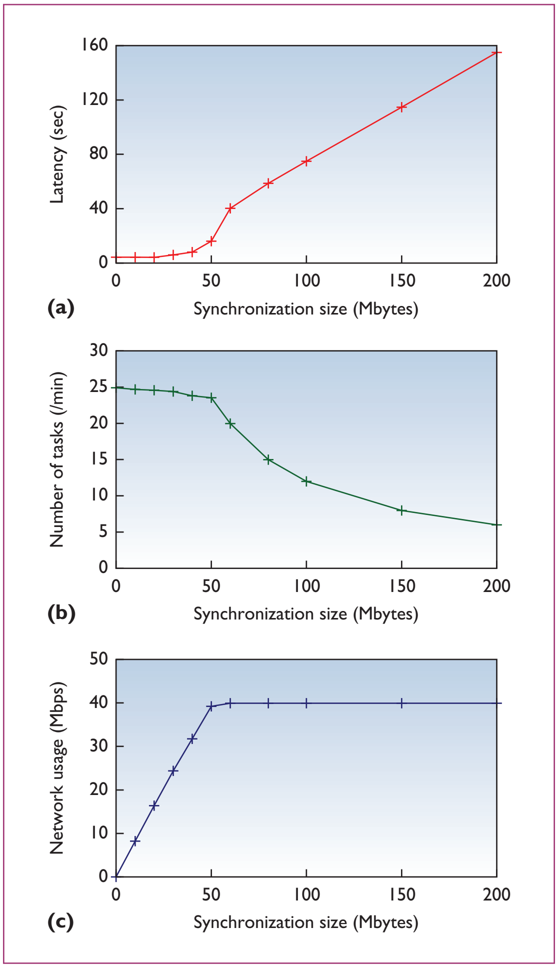


Figure 3. Performance measurements of using fog: (a) latency results, (b) throughput results, and (c) network usage results. From these results, we can see that a smaller synchronization size produces shorter latency, higher throughput, and lower network usage.

Figure 3 illustrates the latency results, the throughput results, and the network usage results of these experiments. From these results, we can see that a smaller synchronization size (that is, n Mbytes) produces shorter latency, higher throughput, and lower network usage. Despite the fact that the synchronization size can't be too small for providing seamless data accessing services, these results demonstrate

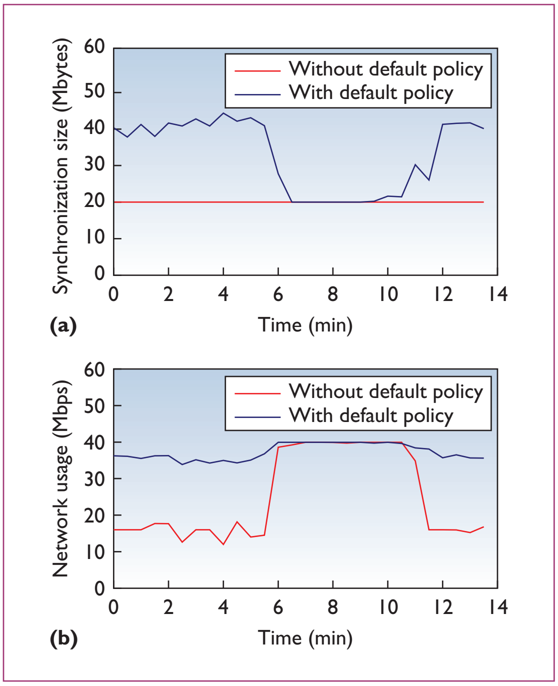


Figure 4. Performance measurements of WM-FOG: (a) synchronization size results and (b) network usage results. When we enable the default synchronization policy, the system makes better use of the network resource, thereby reducing the burden of fully synchronizing the RawVideo tasks in the future.

the benefits of using fog computing in WM-FOG, as tuning the synchronization size is only possible in fog environments.

WM-FOG Performance

WM-FOG makes decisions on how to handle workflow tasks based on the default policies and suggestions from developers. In other words, the default policies are part of the fundamental mechanisms of WM-FOG, which are supposed to provide graceful performance for fog environments.

To evaluate how well our prototype system works, we conduct two experiments. We use only one fog node and the cloud server in these experiments. Once again, we simulate the scenario in which 200-Mbyte RawVideo tasks are handled by our system. A user-specified policy that at least the first 20 Mbytes of rawData should be eagerly synchronized to the cloud is applied to the RawVideo tasks. In the first experiment, we disable the default synchronization policy, so that only the user-specified policy is enforced. In the second experiment, we enable the default synchronization policy, so that the system monitor can monitor the network usage of the fog node. If the system monitor detects that there is a spare network resource between the fog node and the cloud, it informs the cache manager, which in turn tries to synchronize more data for the RawVideo tasks.

Figure 4 illustrates the synchronization size results and the network usage results of these experiments. Clearly, when the default synchronization policy is enabled, the system makes better use of the network resource, and thus the burden of fully synchronizing the RawVideo tasks in the future is reduced. These results demonstrate that WM-FOG is an efficient computing framework for fog environments.

In this article, we compare fog computing and cloud computing in detail, and list a number of research challenges and problems in fog computing. Based on our understanding of these challenges and problems, we propose a software architecture that can incorporate different design choices and user-specified polices flexibly. Then we discuss the design of WM-FOG, a computing framework for fog environments that embraces this software architecture. Evaluation on our prototype system demonstrates that WM-FOG can work effectively and efficiently in fog environments. Future work will involve adding more features to WM-FOG to better serve fog computing applications.

References

1. Sandvine, *Global Internet Phenomena Report*, tech. report, 2015; www.sandvine.com/trends/global-internet-phenomena.
2. F. Bonomi et al., “Fog Computing and Its Role in the Internet of Things,” *Proc. 1st MCC Workshop Mobile Cloud Computing*, 2012, pp. 13–16.
3. L.M. Vaquero and L. Rodero-Merino, “Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing,” *Sigcomm Computer Comm. Rev.*, vol. 44, no. 5, 2014, pp. 27–32.
4. M. Satyanarayanan et al., “Cloudlets: At the Leading Edge of Mobile-Cloud Convergence,” *Proc. 6th Int’l Conf. Mobile Computing, Applications, and Services*, 2014, pp. 1–9.
5. *Mobile-Edge Computing*, European Telecomm. Standards Inst. (ETSI), 2014; www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.

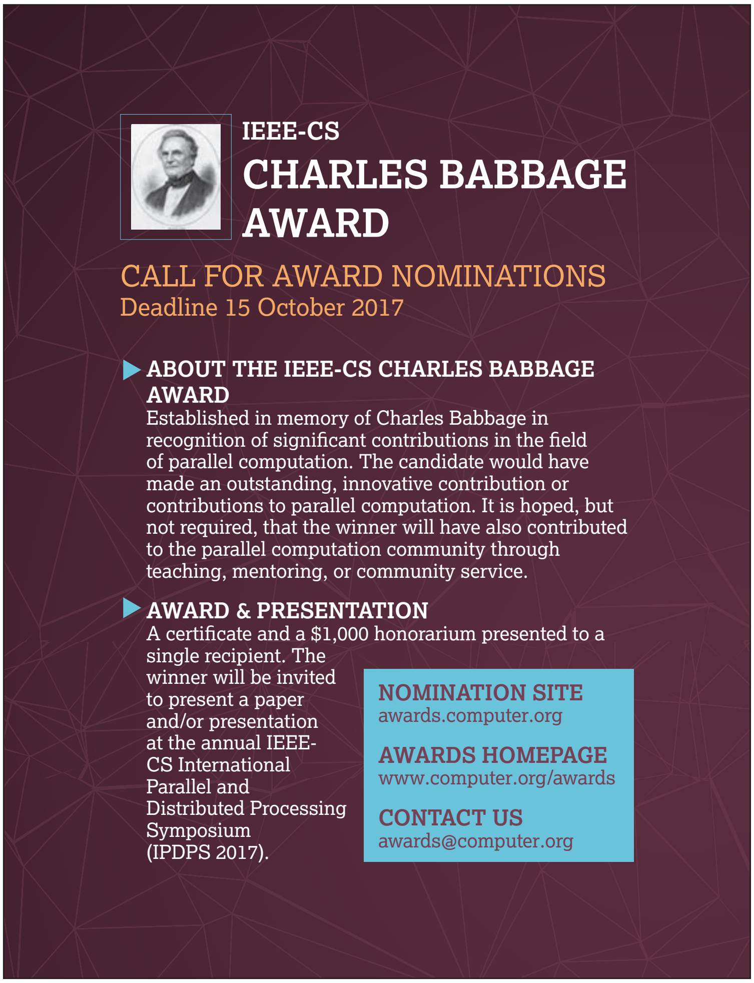
6. N. Fernando, S.W. Loke, and W. Rahayu, "Mobile Cloud Computing: A Survey," *Future Generation Computer Systems*, vol. 29, no. 1, 2013, pp. 84–106.
7. S. Yi et al., "Fog Computing: Platform and Applications," *Proc. 3rd Workshop Hot Topics in Web Systems and Technologies*, 2015, pp. 73–78.
8. Y. Cao, S. Chen, and D. Brown, "Fast: A Fog Computing Assisted Distributed Analytics System to Monitor Fall for Stroke Mitigation," *Proc. IEEE Int'l Conf. Networking, Architecture, and Storage*, 2015, pp. 2–11.
9. T. Zhang, "Fog Boosts Capabilities to Add More Things Securely to the Internet," blog, 3 Mar. 2016; http://blogs.cisco.com/innovation/fog-boosts-capabilities-to-add-more-things-securely-to-the-internet.
10. S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," *Proc. Workshop on Mobile Big Data*, 2015, pp. 37–42.
11. S. Yi, Z. Qin, and Q. Li, "Security and Privacy Issues of Fog Computing: A Survey," *Wireless Algorithms, Systems, and Applications*, LNCS 9204, Springer, 2015, pp. 685–695.
12. Z. Hao and Q. Li, "Edgestore: Integrating Edge Computing into Cloud-Based Storage Systems," *Proc. IEEE/ACM Symp. Edge Computing*, 2016; doi:10.1109/SEC.2016.34.
13. I. Stojmenovic, "Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-to-Machine Networks," *Proc. Australasian Telecomm. Networks and Applications Conf.*, 2014, pp. 117–122.
14. W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 29, no. 5, 2016, pp. 78–81.
15. W. Shi et al., "Edge Computing: Vision and Challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, 2016, pp. 637–646.

Zijiang Hao is a PhD student in computer science at the College of William and Mary. His research interests include mobile-cloud computing, fog/edge computing, geo-distributed storage systems, and consensus algorithms. Hao has an MS in computer science from Tsinghua University, China. Contact him at hebo@cs.wm.edu.

Ed Novak is an assistant professor of computer science at Franklin and Marshall College. His research interests include cybersecurity and privacy on smart mobile devices. Novak has a PhD in computer science from the College of William and Mary. Contact him at enovak@fandm.edu.

Shanhe Yi is a PhD student in computer science at the College of William and Mary. His research interests include mobile/wearable computing and fog/edge computing, with an emphasis on the usability, security, and privacy of applications and systems. Yi has an MS in electrical engineering from Huazhong University of Science and Technology, China. Contact him at syi@cs.wm.edu.

Qun Li is a professor of the Department of Computer Science at the College of William and Mary. His research interests include wireless networks, sensor networks, RFID, pervasive computing systems, and fog/edge computing. Li has a PhD in computer science from Dartmouth College. He received the US National Science Foundation CAREER Award in 2008. Contact him at liqun@cs.wm.edu.



Challenges and Software Architecture for Fog Computing

Hao, Zijiang; Novak, Ed; Yi, Shanhe; Li, Qun

01 Zexin Li Page 1

13/9/2019 9:15

02 Zexin Li Page 1

13/9/2019 9:16

03 Zexin Li Page 1

13/9/2019 9:17

The problems that Cloud have and Fog target to solve.

04 Zexin Li Page 1

13/9/2019 9:15

05 Zexin Li Page 3

14/9/2019 9:03

Fog challenge 1

06 Zexin Li Page 3

14/9/2019 9:03

07 Zexin Li Page 4

14/9/2019 9:03

08 Zexin Li Page 4

14/9/2019 9:04

09 Zexin Li Page 5

14/9/2019 9:06

Fog challenge: security probelms

10 Zexin Li Page 5

14/9/2019 9:06