# Towards a Distributed Computing Framework for Fog

Taeyeol Jeong*, Jaeyoon Chung†, James Won-Ki Hong*, and Sangtae Ha§
*POSTECH, Korea
Email: {dreamerty, jwkhong}@postech.ac.kr
†Princeton University, USA
Email: jaeyoon.chung@princeton.edu
§University of Colorado, Boulder, USA
Email: sangtae.ha@colorado.edu

***Abstract*—Fog computing paradigm has introduced the concept of processing data near the data source. Unlike the cloud, fog computing includes devices with highly varying resources such as heterogeneous computing power, battery, bandwidth, delay, and mobility. The existing distributed computing frameworks, however, have mainly focused on the cloud environment where resources are highly consolidated and stable. This paper presents Crystal, a distributed computing framework for fog. An application consisting of one or multiple Crystal instances offers distributed processing and computing while taking advantage of location transparency, self-healing, auto-scaling and mobility support. Our prototype implementation of MapReduce on Crystal shows benefits of fog computing – fault-tolerant distributed processing over heterogeneous, unreliable, fog nodes while reducing overall latency, thanks to the framework enabling processing close to the data source.**

## I. Introduction

Cloud computing has made a great success and brought the concept of everything-as-a-service (XaaS) model [1], [2]. Fog computing [3], [4], [5] is a new computing paradigm [6], [7], in which computing and processing can leverage nodes closer to where the data is generated (e.g., end-user and edge devices) in the cloud-to-things continuum. As a result, fog has its advantages over cloud because of its proximity to the 'ground' [8]:

- **Security:** Exploiting proximity of fog resources closer to users reduces a chance of data breach.
- **Cognition:** Awareness of client-centric objectives ensures a better understanding of user demand.
- **Agility:** Fog supports a rapid innovation and scale using available fog resources in the cloud-to-things continuum.
- **Latency:** Leveraging fog resources closer to users provides low and predictable latency to data processing applications.
- **Efficiency:** Fog computing reduces the overhead of the cloud and the network core by processing data locally.

The existing distributed computing / processing frameworks in the cloud, such as MapReduce [9], however, are not directly applicable to fog [10], [11]; the cloud has mostly homogeneous computing nodes, well-structured network topology, and reliable network connectivity, while the fog has to deal with the other extremes, such as heterogeneous computing nodes with high churn rates and unstructured networks with frequent disruptions.

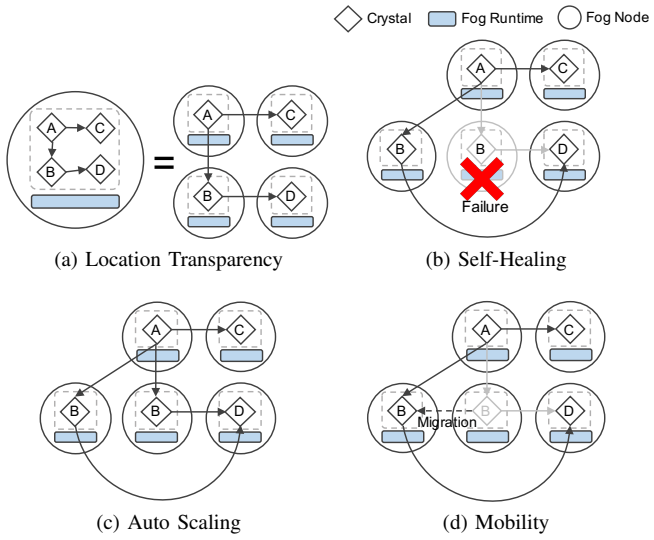
This paper proposes *Crystal*, a distributed computing / processing framework for fog. Crystal provides an easy abstraction for fog application development while supporting location-transparency, self-healing, auto-scaling and mobility support. The proof-of-concept implementation of Crystal is evaluated by designing a MapReduce framework for fog. The performance results show that MapReduce on Fog, compared to Cloud, gives the benefit of reduced completion time and power consumption, thanks to its computing / processing close to the data source.

## II. Design Considerations

Developing a fault-tolerant fog application spanning over fog nodes requires high programming complexity and dealing with all the exceptions and failures, inevitably leading to complicated application development process. To tackle these issues, the proposed fog computing framework Crystal provides an easy abstraction for fog application development. The name Crystal is derived from ice fog, a type of fog composed of tiny ice crystals suspended in the air. Similarly, in Crystal framework, a fog application is composed of multiple components called Crystals spanning over fog nodes. Crystals are loosely-coupled components, each of which can run a standalone component, and they communicate each other through message passing. The functional definition of a Crystal component can vary from function-level to microservice-level depending on its roles in a fog application.

The proposed Crystal framework follows *'let it disappear'* philosophy to reflect the inherently unreliable nature of fog where any node can leave and join at any moment. Whenever a fog node disappears from the fog by any reason, a fog application consisting of Crystals automatically heals by exploiting available fog and/or cloud nodes. Thus, developers can use Crystals as building blocks for fog applications without concerning failures of fog nodes. A fog application using Crystals can take full advantage of location transparency,

978-1-5386-3666-4/17/$31.00 ©2017 IEEE




(a) Location Transparency (b) Self-Healing

(c) Auto Scaling (d) Mobility

Fig. 1. Crystal Design Properties

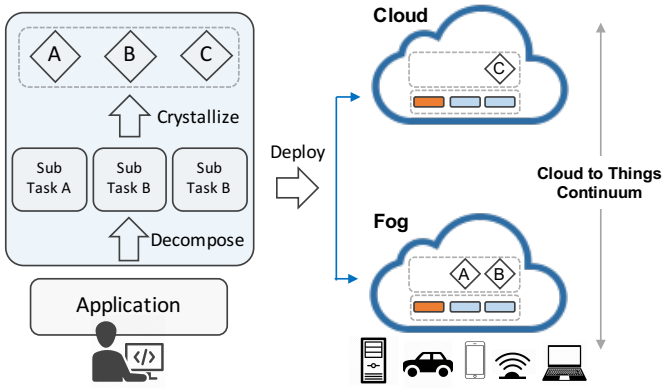



Fig. 2. Overview of Fog Computing with Crystal

self-healing, auto-scaling and mobility support (Figure 1) described below:

**Location Transparency:** Crystal components should be able to run on any node at any time. While the location of each Crystal instance may affect the overall performance of a fog application, it should not affect the context of the application. The concept of Crystal's location transparency enables a fog application to be easily decomposed and deployed over a pool of fog nodes. Also, location transparency gives developers an illusion that a fog application development is more like a local application development.

**Self-Healing:** Crystal components should embed a self-healing functionality to survive in fog environment. Fog's highly unreliable and unpredictable nature imposes a significant burden on developers to deal with all the exception and error handlings. Crystal assumes that any node can fail at any moment in fog. In that sense, Crystal components keep monitoring and respawning each other for self-healing, rather than expecting developers to program an error-free fog application. A fog application can be presented as a directed acyclic graph according to the communication patterns of its Crystals. When a parent Crystal detects a failure from its child Crystal instance, it automatically re-creates it on another fog node. All of these procedures are transparent to the user application, and also to the user code.

**Automatic Scaling:** Crystal components should automatically scale out and scale in when a Crystal instance is busy or idle, respectively. In case of a stateless Crystal instance, Crystals are seamlessly scaled out and scaled in by creating or killing replicated Crystal instances. Messages to the replicated Crystal instances are automatically load-balanced. In case of a stateful Crystal instance, its state is written to a distributed data store and accessed by replicated Crystal instances.

**Mobility Support:** Crystal components should move from one fog node to another when migration is required. With Crystal's mobility, a fog application consisting of Crystals can dynamically expand itself from things, to the fog, to the cloud, and also vice versa. When a Crystal instance of a fog application does not meet the minimum running requirements anymore (e.g., lack of resources, or location change), it finds an adequate fog node and migrates to the node. Crystals are responsible for delivering the messages designated to the migrated Crystal instance and it is transparent to the user code.

## III. Crystal: A Fog Computing Framework

Figure 2 shows the overall architecture of Crystal-based fog computing. When a fog application is initiated, it is decomposed into subtasks by the fog agent. The fog agent then transforms the subtasks into standalone and deployable Crystal instances. To deploy these Crystal instances, the fog agent first sends a resource request to a fog tracker which tracks the list of fog nodes and their resource availability. According to each Crystal instance's resource requirements, the fog tracker sends back the list of most adequate fog nodes.

### A. Fog Agent

A fog agent is a lightweight software runtime running on a fog node. Fog agents register themselves to a fog tracker and update their resource status. By cooperating each other, geographically close fog agents build up a fog cluster. Fog agent is the key enabler which provides location transparency, self-healing, automatic scaling and mobility properties to Crystal instances.

A fog agent is in charge of decomposition, crystallization (the process of transforming subtasks into standalone and deployable Crystal instances), and deployment of a fog application. When an application is initiated on a fog node, it first decomposes the application into subtasks. Decomposition can be performed in both automatic and manual approach. In the automatic decomposition process, the application needs to follow an object-oriented programming model. To make each object a standalone component, each object is wrapped by a new main function, and the communication between objects

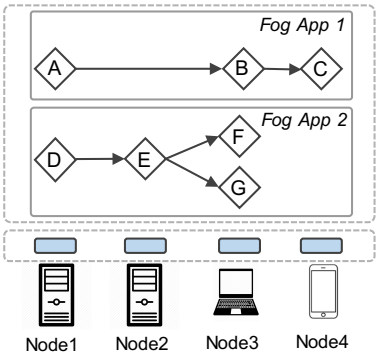
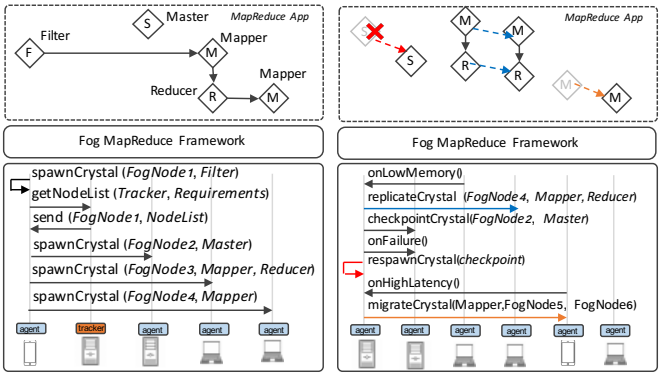



Fig. 3. DAG Representation of Fog Applications




(a) MapReduce App Deployment (b) MapReduce App Scheduling

Fig. 4. Fog MapReduce Sequence Diagram with Fog APIs

is replaced by remote procedure calls or message passing. In this approach, the application is decomposed into object-level. Each object becomes a subtask which later becomes a Crystal. In the manual approach developers are expected to decompose an application into subtasks.

To reduce the burden on the fog application developers, an automatic orchestration is provided by fog agents. When a fog application is deployed over fog nodes, the fog application can be described as a directed acyclic graph (DAG) of Crystals. An example DAG representation of the fog application is shown in Figure 3. When a fog agent is hosting a Crystal instance the fog agent is responsible for taking care of the Crystal's child Crystal instances. A fog agent monitors its child Crystal instances on its child fog agents. When a child Crystal instance fails, the parent fog agent immediately respawns the failed Crystal instance. For example, when Node 3 fails, the fog agent on Node 1 respawns Crystal B (*Fog App 1*), and the fog agent on Node 2 respawns Crystal F and G (*Fog App 2*). Also, a parent agent performs scale out or migration when a child Crystal instance suffers from insufficient resources, or when the fog node does not meet the Crystal instance's resource requirements anymore.

### B. Fog Tracker

To easily discover available fog resources, we borrow the concept of trackers in peer to peer networks. A fog tracker is a node which keeps tracks of the list of fog nodes and their available resource information, similar to BitTorrent's tracker server [12], [13]. Fog resource information in a fog tracker includes CPU, RAM, Storage, operating system, mobility, battery, location information, etc. When a fog runtime (fog agent) is initialized on a fog node, it registers itself to a fog tracker and periodically sends health-check messages to the fog tracker. When fog resources are queried by a fog agent, the fog tracker first looks at the application's requirement and finds out the most appropriate fog resources. The selected fog resources will be notified to the fog agent who requested the resource. Ideally, a fog tracker is expected to keep track of nodes in a fog cluster. For example, a campus fog tracker can build a campus-wide fog cluster, and let fog applications leverage fog resources in the campus.

The proposed fog tracker system supports tracker-to-tracker communication for the sharing of peer list among fog clusters. A fog tracker shares its fog node list to other fog trackers. That is, tracker-to-tracker communication eventually enables a fog node to see the global view of fog resources. A fog tracker in the cloud with a cooperation of the existing resource managers (i.e. YARN [14] and Mesos [15]) makes it possible for fog agents to leverage cloud resources. In this way, fog node not only utilizes the resources in the fog cluster it belongs to, but also exploits resources from any other fog clusters (*fog-to-fog federation*) or cloud clusters (*fog-to-cloud federation*). A hierarchical tracker (hierarchical fog cluster) architecture can be used for better scalability.

A fog tracker also provides a naming system for both fog nodes and Crystal instances. For location transparency and mobility support, Crystal instances are referred as unique names, not by its physical addresses. This naming system separates the physical location of a Crystal instance, from its identifier. A Crystal instance's name does not change, even if its actual location changes. When it moves to another fog node, by updating its new location information in the tracker, other Crystal instances can still communicate with the Crystal.

### C. Programming Interfaces

Table I and II present fog programming interfaces and fog event handlers, respectively. Even though Crystal components support self-healing, automatic scaling, and seamless mobility under the hood, developers can still implement explicit failover, scaling, and migration functionalities to provide fine-grained orchestration for their Crystal components.

## IV. Application: Fog MapReduce

As an application leveraging the proposed Crystal framework, we present *Fog MapReduce*, a distributed data processing framework for fog environment. The main goal of Fog MapReduce is to support MapReduce applications over fog nodes in order to process data closer to where it is

TABLE I
FOG PROGRAMMING INTERFACES

| Operation | Description |
|---|---|
| send | Send a message to a Crystal |
| receive | Receive a message and process it |
| stayInCloud | Crystal will stay in the cloud |
| stayInFog | Crystal will stay in the local fog cluster |
| stayInThing | Crystal will stay in the client node where it was created |
| moveToCloud | Crystal will move to the cloud |
| moveToFog | Crystal will move to the fog |
| moveToThing | Crystal will move to the client node |
| replicateCrystal | Scale out a Crystal and do load balancing |
| spawnCrystal | Create a Crystal instance on a node |
| migrateCrystal | Migrate a Crystal instance to a node |
| checkpointCrystal | Save the state of Crystal |
| respawnCrystal | Re-create a Crystal from a checkpoint |
| killCrystal | Terminate Crystal instance |
| getCrystal | Get name of Crystal |
| getCrystalList | Get the list of Crystals for an application |
| getCrystalParent | Get the list of Crystal's parent Crystals |
| getCrystalChild | Get the list of Crystal's child Crystals |
| getCrystalLocation | Get location information of a Crystal |
| getCrystalLatency | Get latency from a Crystal to another |
| getNode | Get name of a fog node |
| getNodeList | Get the list of fog nodes in the fog cluster |
| getNodeLocation | Get location information of a node |
| getNodeLatency | Get latency from Crystal to a fog node |
| getFog | Get name of the local fog cluster |
| getFogList | Get the list of fog clusters |
| getFogLocation | Get location coverage of a fog cluster |
| getFogLatency | Get latency from Crystal to a fog cluster |
| setLatency | Set latency limit a Crystal has to guarantee |
| setStorage | Set storage limit a Crystal can use |
| setMemory | Set memory limit a Crystal can use |
| setCPU | Set CPU limit a Crystal can use |

TABLE II
FOG EVENT HANDLERS

| Event | Description |
|---|---|
| onCreate | New Crystal created |
| onFailure | Crystal failed |
| onKill | Crystal killed |
| onMigrate | Crystal migrated |
| onReplicate | Crystal replicated |
| onHighLatency | Average Latency is higher than limit |
| onLowCPU | Node running out of CPU |
| onLowMemory | Node running out of memory |
| onLowStorage | Node running out of storage |

generated. Processing data near the data source brings a lower response time and a better privacy for sensitive data. However, fog nodes are likely to join and leave frequently during task execution. In that sense, Fog MapReduce is designed to fully utilize the proximity of fog nodes for data processing while dealing with unreliability. Existing MapReduce frameworks adopt a centralized master-worker architecture. However, this architecture does not consider or deal with failures of the master because the master-worker architecture is expected to run on a reliable computing node. When the master fails, workers are not able to communicate with the master anymore, which results in the failure of the whole MapReduce application. On the other hand, the master and workers in the Fog MapReduce are implemented by Crystals components. Both the master and workers inherit Crystal's properties so that a node failure does not stop

```
//MapReduce source code of main.go

package main

import "fog/mr"

func main() {
    mr.New().OpenFromText("data_source").
    Filter(func(line string) bool {
        return strings.HasPrefix(line, "[VALID]")
    }).
    Map(func(key string) int {
        return 1
    }).
    Reduce(func(x int, y int) int {
        return x + y
    }).
    Map(func(x int) int {
        return x
    }).
    Run()
}
```

a MapReduce application. Figure 4 describes how a Fog MapReduce application works. Whenever a user MapReduce application is initiated, a new master Crystal instance is spawned on a fog node. This master then starts to perform job scheduling and spawn MapReduce-related Crystals such as mappers and reducers to other fog nodes. The MapReduce Crystal instances then process data in a distributed way and the final result is sent back to the collector Crystal. The result can be sent to a cloud storage for availability or future analysis. A Fog MapReduce word count example code is described above.

## V. IMPLEMENTATION

This section describes implementation details of the prototype Crystal fog computing framework. The overall system is written in Golang. To implement Crystal that can run on heterogeneous fog nodes, each Crystal component is containerized before deployed over fog nodes. Containerizing a Crystal instance also achieves isolation of each Crystal from others. For the containerization, the de-facto industry standard Docker is used. To make the task decomposition process faster, containerization is done by using a minimized base image with around 4MB. For those platforms not supported by Docker (e.g., IoT and mobile platforms), fog agents cross-compile those Crystals for the target platforms.

For the message delivery among Crystals, and also among fog agents, *nats* distributed message queue is modified. Because of the unreliability of fog environment, successfully delivering a message between two fog nodes is not always easy. Also, many mobile fog nodes are located behind Network Address Translation (NAT), which makes direct communication between two mobile nodes even more tricky. To deal with the guaranteed message delivery and NAT traversal, the proposed framework exploits some strong fog nodes as message brokers. Those strong fog nodes have relatively reliable network connectivity, less mobility, and sufficient

computing resource. This message brokers guarantee that messages are fault-tolerant and at-least-once delivered.

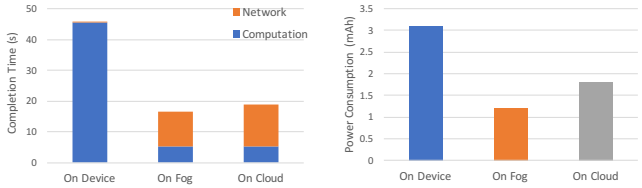
Fog tracker is implemented on top of distributed key-value store *etcd* to avoid a single point of failure. This distributed key-value store keeps track of fog nodes and their resource information. The naming system is also implemented with the distributed key-value store. It keeps the match between a name of each Crystal and the corresponding location.

## VI. Evaluation

Evaluation of the prototype is performed on a mobile device, three fog nodes, and three cloud nodes. For each fog node, a VM equipped with a Quad-core Intel Xeon CPU E5-2697 at 2.70GHz and 4GB RAM, running Ubuntu 16.04.1, is used. For each cloud node, an Amazon AWS EC2 m4.large (US East, Northern Virginia) server equipped with 6.5 EC2 Compute Units with 8GB RAM, running Ubuntu 16.04 LTS, is used. For the mobile device, a Samsung Galaxy S6 equipped with Octa-core CPU (4x2.1 GHz ARM Cortex-A57 & 4x1.5 GHz ARM Cortex-A53) and 3GB RAM, running Android 6.0.1, is used. The mobile device is connected to Princeton University's campus Wi-Fi network during the experiments.

Figure 5 shows performance evaluation of application decomposition using Crystal. Figure 5a evaluates the completion time of a word frequency count with a 30MB text file. When the job is processed only on the mobile device, it takes for around 45s to finish the task. When the task is decomposed to the fog, the completion time is reduced by more than 63%. When the task is decomposed to the cloud, the completion time is reduced by 58%. Computation time out of the total completion time is 5.420s for the fog, and 5.184s for the cloud. Network time out of the completion time is 11.040s for the fog, and 13.741s for the cloud. Figure 5b measures the power consumption of the mobile device during the word frequency count. When the task is processed only on the mobile device, it consumes 3.1 mAh, but the power consumption is decreased to 1.2 mAh when the task is decomposed to the fog. When the task is decomposed to the cloud, it takes 1.8 mAh of power consumption. This result shows that fog environment can provide reasonable computing capability with lower network latency compared to the cloud. Also, it indicates that utilizing fog resources can increase the computing / processing performance of IoT and mobile applications, and also reduces their power consumptions.

Figure 6 shows performance comparison between Fog MapReduce and Apache Spark 1.6.0. A MapReduce word frequency count job is executed over three fog nodes because Spark does not support heterogeneous nodes such as IoT and mobile devices. Both Spark and Fog MapReduce are running on Docker containers for the fair comparison. Data for MapReduce job is assumed to be distributed on the nodes. Fog MapReduce still shows similar completion time compared to Spark (Figure 6), although it is designed to work on highly heterogeneous and unreliable nodes.




(a) Completion Time (b) Power Consumption

Fig. 5. Fog App Decomposition Performance

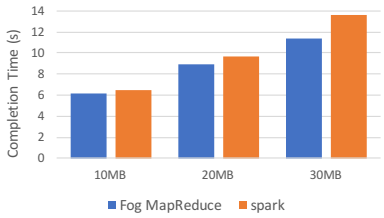



Fig. 6. MapReduce Performance Comparison

## VII. Related Work

Crystal's location transparency is inspired by the actor programming model [16], [17], and its self-healing property is inspired by various distributed computing systems [18], [19], [20], [21]. For mobility support, the concept of locator/identifier separation [22], [23], [24] is adopted. Programming models for mobile applications were studied in [25], [26].

The design of a fog tracker is partially inspired by the concept of BitTorrent's tracker functionality [12], [13]. Widespread adoption of peer to peer networks has proven the effectiveness of tracker nodes that keep track of the list of peer nodes.

Many advanced data processing frameworks have been introduced after the success of MapReduce [9], but those frameworks are designed and expected to run on reliable computing environment (e.g., cloud). Fog MapReduce provides MapReduce over unreliable computing nodes [27] and realizes data processing near the data source.

## VIII. Conclusion

This paper proposes Crystal, a distributed computing framework for fog applications. Its *'let it disappear'* philosophy frees developers from struggling with exception and error handlings for highly unreliable fog environment. Crystal allows developers to easily build a sustainable fog application by using a Crystal instance as a building block. Fog applications made up of Crystals can fully support location transparency, self-healing, auto-scaling, and mobility, which significantly reduces programming complexity. With the proposed Crystal framework, applications can leverage fog resources along the cloud to things continuum by allowing decomposed components of an application to reside across the things, fog, and cloud.

ACKNOWLEDGEMENT

This material is based upon work supported by the Defense Advanced Research Projects Agency under Contract No. HR001117C0052. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This work was partially supported by the NSF under Grant CNS-1525435. This work is also partially supported by Comcast Innovation Fund Research Grant. This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2017-0-01633) supervised by the IITP (Institute for Information & communications Technology Promotion).

## REFERENCES

[1] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, B. Huberman, J. Manley, C. Patel, P. Ranganathan, and A. Veitch, "Everything as a service: Powering the new information economy," *Computer*, vol. 44, no. 3, pp. 36–43, 2011.

[2] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (xaas) on the cloud: origins, current and future trends," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 621–628.

[3] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, pp. 169–186.

[4] "Openfog architecture overview," White Paper, OpenFog Consortium Architecture Working Group, Feb. 2016, https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf.

[5] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[6] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing," in *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on*. IEEE, 2014, pp. 325–329.

[7] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[8] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*. IEEE, 2014, pp. 117–122.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[11] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[12] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *ACM SIGCOMM computer communication review*, vol. 34, no. 4. ACM, 2004, pp. 367–378.

[13] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *International Workshop on Peer-to-Peer Systems*. Springer, 2005, pp. 205–216.

[14] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[16] C. Hewitt, "Viewing control structures as patterns of passing messages," *Artificial intelligence*, vol. 8, no. 3, pp. 323–364, 1977.

[17] J. Armstrong, R. Virding, C. Wikström, and M. Williams, "Concurrent programming in erlang," 1993.

[18] S. Bykov, A. Geller, G. Kliot, J. R. Larus, R. Pandya, and J. Thelin, "Orleans: cloud computing for everyone," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 16.

[19] P. A. Bernstein, S. Bykov, A. Geller, G. Kliot, and J. Thelin, "Orleans: Distributed virtual actors for programmability and scalability," *MSR-TR-2014–41*, 2014.

[20] V. Vernon, *Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka*. Addison-Wesley Professional, 2015.

[21] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[22] D. Farinacci, D. Lewis, D. Meyer, and V. Fuller, "The locator/id separation protocol (lisp)," 2013.

[23] D. Farinacci, V. Fuller, D. Lewis, and D. Meyer, "Lisp mobile node," *Work in Progress*, 2011.

[24] T. Jeong, J. Li, J. Hyun, J.-H. Yoo, and J. W.-K. Hong, "Lisp controller: a centralized lisp management system for isp networks," *International Journal of Network Management*, vol. 25, no. 6, pp. 507–525, 2015.

[25] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM, 2013, pp. 15–20.

[26] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy, "Customizable and extensible deployment for mobile/cloud applications." in *OSDI*, 2014, pp. 97–112.

[27] F. Marozzo, D. Talia, and P. Trunfio, "P2p-mapreduce: Parallel data processing in dynamic cloud environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1382–1402, 2012.