Cresco: A distributed agent-based edge computing framework

V. K. Cody Bumgardner
Department of Computer Science
University of Kentucky
Lexington, Kentucky, USA
Email: cody@uky.edu

Victor W. Marek
Department of Computer Science
University of Kentucky
Lexington, Kentucky, USA
Email: marek@cs.uky.edu

Caylin D. Hickey
Department of Computer Science
University of Kentucky
Lexington, Kentucky, USA
Email: caylin.hickey@uky.edu

***Abstract*—The Cresco distributed agent-based framework is designed to address the challenges of edge computing. We present an actor-model implementation for the management of large numbers of geographically distributed services, comprised from heterogeneous resources and communication protocols, in support of low-latency realtime streaming applications.**

We present the purpose of our work, the basic methodology, the initial results already obtained, the relationship to the existing software, and the potential of the presently implemented framework to a number of potential further projects that could be developed on the basis of the existing implementation.

I. Introduction

In China alone there were a reported 9 billion devices as of 2014, with estimates of 24 billion by 2020 [1]. By the year 2020 there will be an estimated 50 billion network connected devices globally [2], [3].

Whether referred to as IoT, Cyber-Physical Systems (CPS) [4], Machine to Machine (M2M) [5] technologies, Industrial Internet [6], or Smart Cities [7], all of these efforts aim to improve society through the harnessing of data and resources from distributed system.

The Cresco framework was created to assist in the development of globally distributed applications where data collection and processing take place on the network edge. In addition, the framework provides the ability to coordinate further processing of data between other edge and remote data center resources. For example, the framework can be used to develop applications that process high-rate or globally inaccessible data on network edges, assign workloads to and between appropriate edge devices, and coordinate central processing of filtered, enriched, and/or edge-aggregated data.

The authors believe the Cresco framework addresses the following characteristic challenges inherent to edge computing, as defined by Bonomi et al., 2012 [8]:

- *Low latency and location awareness*: There is a tradeoff between moving data to processing resources or moving resource to sources of data generation. The Cresco framework provides a global view of distributed resource performance allowing for the programatic management of both low latency network edge processing and data center processing. The Cresco *Agents*, described in Section II-B, and *Agent-Plugins*, described Section II-C, operate in a role-based structure where workload assignments and locations are known globally.
- *Wide-spread geographical distribution*: The Cresco framework was developed to operate on a globally distributed level. Using our hierarchical agent architecture, described in Section II-D, *Cresco-Controller*, we provide fine-grained control over the structure, communication protocols, and security of distributed systems.
- *Very large number of nodes*: *Actor-model* [9] concurrency provides great scalability through a natural hierarchical structure of node processes. In order to manage large numbers of nodes, higher levels in the hierarchical structure are responsible for lower levels in the structure. An Actor implemented in an Agent-Plugin is managed by an Agent, which is managed by a hierarchy of agent controllers.
- *Mobility*: In our framework Agents are deployed directly on mobile devices or Agent-Plugins can be used to represent individual external devices or networks of devices. Our text-based messaging protocol allows Cresco participating *Actors* described in Section II-A, *Methodology* to be implemented in many languages or directly in hardware.
- *Predominant role of wireless access*: Cresco Agents can be deployed in wireless agents and participate as part of a wireless network. Additionally, Cresco can be used to construct distributed applications using common, lightweight wireless protocols like MQTT [10].
- *Strong presence of streaming and real-time applications*: Cresco originated from the need to deploy interconnected, but geographically distributed resources for streaming and real-time applications. Using Cresco, large distributed applications can be deployed for real-time event processing and data enrichment, cross-region aggregation, and central stream analysis.
- *Heterogeneity*: A key differentiator of the Cresco framework its ability interface with heterogeneous computational, network, and operating environments through text-based configurations and messaging.

978-3-901882-85-2

The Cresco framework, the subject of this paper, has been designed from the ground up with edge computing applications in mind. The following section describes the framework architecture.

II. Architecture

The Cresco framework was created to address the challenges of managing resources across local, regional, and global domains. Cresco [11] is a free and open-source distributed agent-based resource management framework available under the Apache Version 2.0 license [12]. Initial project goals are to provide a framework for resource comparison, performance observation, and establish a means of control over collections of resources and related applications.

A. Methodology

The presented framework is based on Actor-model distributed concurrency. In this model an *Actor* is a primitive unit of isolated computation that uses asynchronous messaging to communicate with other Actors. While the details of the Actor-model are beyond the scope of this paper, basic operations of Actors include message-based creation of more Actors, Actor-to-Actor messaging, or the generation of state decisions applying next arriving message. Erlang [13], an example of a popular programming language based on the Actor-model, introduced a "let it crash" philosophy for distributed computation. Instead of focusing on defensive programming to prevent failures, using an offensive (create, monitor, and verify) philosophy one relies on Actors to supervise other Actors creating "self-healing" distributed processing environments. In addition, the isolated operation of Actors makes continuous self and supervisor reporting of KPIs across heterogeneous environments possible.

Actors operate in isolation with the exception of inter-Actor messages, making Actor communication critically important, especially for geographic distribution. The Cresco Actor-model implementation aims to address challenges related to interoperability, performance, and security related to Actor communication channels.

While Actors are typically represented as critical sections of code in programming frameworks like Erlang, Scala [14], and Akka [15], Cresco Actors can be critical sections of code executing from within the native (Java) framework or abstracted Actor interfaces (APIs, CLIs, etc.) into external systems. For instance, a Cresco Actor responsible for aggregating data within the framework communicates with other Cresco Actors responsible for monitoring, measurement, and processing of an external sensor networks. In support of heterogeneous operations across a wide variety of environments both Actor configurations and messages are UTF-8 [16] encoded text. It is sufficient to think of the Cresco Actor-model implementation as a graph of text-defined primitives describing distributed applications.

The text-based abstraction of Cresco Actors is accomplished through the implementation of software agents. Agent terminology makes it easier to solve the problems related to programatic information processing and resource management. We are not developing agent programming, which is an existing large area in the methodology of programming, but rather use an existing framework presented by Subrahmanian et al., 2000, [17] as part of this effort. Cresco software *Agents* and *Agent-Plugins* have been developed in the Java language, but given the text-based nature of our Actor implementation agents can be developed for a wide variety of hardware and software environments.

Actors are naturally hierarchical with every Actor being a child of another Actor. In order to manage the communication aspects of the Actor hierarchy in our agent system we have defined three primary operational agent hierarchies including *Agent*, *Regional*, and *Global*. While additional levels of Actor hierarchies may exist within the defined agent hierarchies, agent discovery, communication, and security isolation is enforced within each level. In addition, working with agents in a hierarchical model makes it easier to solve issues related to system scalability.

Cresco is composed of three primary types of components:

- *Cresco-Agent*: Endpoint resource management unit in charge of orchestrating *Cresco-Plugins* state, messaging, and monitoring aggregation.
- *Cresco-Plugins*: Work units providing communication channels at the agent, region, and global level, performance monitoring, as well as custom work units through extension of the Cresco-Plugin-Library.
- *Cresco-Controller*: Special plugin required to establish agent, regional, and global management plane.

The following subsections will further define each of these component types in the context of the Cresco framework.

B. Cresco-Agent

Agent-based modeling (ABM) [18] is a computational model used in the simulation of agent interaction. There exist a large body of research for ABM across many disciplines, including: biology, economics, social sciences, and engineering.

On a high-level, the Cresco framework functions as an Actor-model, implemented as a multi-agent system. The Cresco-Agent provides dynamic configuration, loading, and unloading of Cresco-Agent-Plugins within the agent. The agent directly routes messages between intra-agent plugins and with the use of the *Cresco Agent Controller Plugin*, described in Section II-D, *Cresco Controller*, transmit messages on regional and global levels. The Cresco-Agent provides the underling runtime environment for all native Cresco components.

C. Cresco-Plugins

Cresco plugins provide communication channels for agents, native execution environments, interfaces to resources, and other operational functions. There are two classifications of plugins in the Cresco framework. The first type is a resource plugin, and the second type is used in the core operation of Cresco. *Resource plugins* must be implemented for specific

resource(s) managed by the Cresco framework. These plugins report resource information pertaining to active resource instances. For example, the same *resource plugin* deployed on agents across two providers might provide differing performance information, for the same observed workload. *Core plugins* are required by the Cresco framework. We discuss one such plugin, the *Cresco-Controller*, in the following subsection.

D. Cresco-Controller

The *Cresco Controller* is a core plugin required by an agent to govern its operation. On agent initialization, the controller plugin is loaded and begins its discovery of other Cresco controllers, deciding its *operating mode* based on local discovery and configuration parameters provided by the Cresco Agent. Following a secure initial discovery phase[1], in which secure connection credentials are generated for each Cresco Agent, the open-source package ActiveMQ [19] is used for further communications between agents.

By default, ActiveMQ uses auto wire format detection AUTO [20] to detect the protocol being used for communications in the Cresco framework. Natively, ActiveMQ supports channels using MQ telemetry transport (MQTT[2]), advanced message queuing protocol (AMQP) [21], OpenWire [22], representational state transfer (REST) [23], RSS and Atom [24], streaming text oriented messaging protocol (STOMP) [25], web services invocation framework (WSIF) [26], WebSocket Notifications [27], and extensible message and presence protocol (XMPP) [28] natively. Regardless of the transport protocol, the text-based MsgEvent format is used. The MsgEvent protocol allows for the heterogeneous operation of the Cresco framework across a broad variety of communication channels. Contained within the MsgEvent object are attributes related to the state and behavior. This is achieved through a combination of required header fields, used in the routing of messages throughout the Cresco Framework, and a set of optional supplementary fields, consisting primarily of key-value pairs, constituting the body of the message.

An *operating mode* defines where a controller resides in the Cresco hierarchy. There are three operating modes of a controller, shown below:

- *Global*: Responsible for establishment of a global management plane across and inter-region message routing.
- *Regional*: Responsible for communication between regions and with one or more global controllers as well as intra-regional message routing.
- *Agent*: Responsible for communication with regional controller.

1) Global Controllers: The global controller establishes a global view of resource status across regions. This view is inclusive of all resources assigned at regional and agent levels. A global resource view is established by maintaining a graph database of all known local and regional relationships. We use the open-source graph database OrientDB [29] for database services. We define all node and edges in our graph database to be the *complete graph* of the Cresco system. We define the *topology graph* to be a sub-graph of the complete graph, which contains nodes, edges, and labels related to arrangement and connectivity of Cresco components. We define arrangement in this context to be the spacial positioning of Cresco components based on regional, agent, and plugin assignments. We define connectivity in this context to be the path in which MsgEvent messages are routed between components. The topology graph is maintained by the global controller, based on regional topology information provided by the regional controller.

We define the *resource graph* to be a sub-graph of the complete graph of the system, which contains nodes, edges, and labels related to the arrangement, configuration, and utilization of resources managed by Cresco components. In this context, configuration is related to the textual configuration of agents and plugins as described in Section II-B, *Cresco Agent*. Utilization is defined as a set of resource-specific metrics, reported by resource managing Cresco plugins.

The resource graph includes active plugins related to specific global applications. Information is provided to the controller by one or more regional controllers. These plugins establish communication with the global controller and provide both regional and agent resource metrics for topology and resource graphs.

2) Regional Controllers: The regional controller plugin manages reachable agents in its region. This plugin serves as a gateway connecting reachable intra-regional components and inter-regional components. Agents status is detected by the regional controller plugin as changes on the agent-level are discovered. However, it is not necessary for the agent to maintain connectivity to a Regional controller at all times. The system is designed with the expectation that agents and plugins can both appear and disappear without warning, so state discovery is accomplished using several methods described in the following subsection.

3) Agent Controllers: The agent controller manages inter-agent communications to the regional controller.

III. Operations

A. Agent Initialization

On startup, the Cresco Agent validates its configuration files and will attempt to establish regional communication using the *Cresco-Controller*. An unconfigured Controller Plugin will utilize automatic local discovery over both IPv4 broadcast and IPv6 multicast channels to search for existing regional controllers with which to communicate. Discovery functions as follows:

When attempting to connect to a Regional Controller, as well as a Global Controller, an encrypted shared secret is checked to ensure the joining Cresco Agent has the proper rights to join the regional group. As an added benefit, the formation of Regional and Global controlled groups can be shaped by controlling the shared secret they use to join the group. The automatic discovery can also be overridden through

[1]The initial Discovery phase functions over IPv6, in addition to IPv4.

[2]MQTT is a widely adopted transport protocol in IoT applications.

configuration such that the Cresco Agent will alway elect to be a regional controller, should it be desirable to select which host controls regional communications.

The regional controller acts as a gateway between local agents inside its region and the Global Controller with which it communicates. This abstracts away any need for lower level agents to have any knowledge of the presence of the Global Controller. These channels are monitored to ensure connectivity. Should a Regional Controller fail in a specific region, Local agents will detect the failure and attempt to connect to another Regional controller through the previously detailed discovery functions, or elect to become a Regional Controller in the absence of a suitable candidate. Once the communications channel is established, the remainder of the plugins configured to load automatically are processed and loaded by the Cresco Agent. In the next section, we will cover the basics of Cresco Plugins as well as how they are accessed via the Cresco Agent.

B. Plugin Engine

Plugins are agent modules that are implemented through independent compilation units. Loading and unloading of the plugin codebase is performed at runtime by the host agent, which loads the code through independent and contained systems within the virtual memory space of the agent, with the agent controlling the access levels granted to the plugin code. In order to facilitate communication between agents and plugins, a common *CPlugin Abstract* definition is shared between their respective codebases. The plugin is then required to implement a small subset of these methods from the abstract, with the ability to override more low-level operations should the need arise.

C. Message Routing Engine

The agent's msgInQueue is a concurrently accessible FIFO (first-in-first-out) ordered MsgEvent queue. The *MsgInQueue* process thread waits for the the arrival of messages in the queue. For each new message arrival a *MsgRoute* thread is created to route the incoming message and the message is removed from the msgInQueue.

Messages are either routed to a specific plugin or executed on the agent itself. The route thread terminates on route completion, which is sufficient for unidirectional messages. However, there are cases where we want a response to our messages.

In the context of the Cresco framework we define Remote Procedure Calls (RPC) as bi-directional asynchronous method executions. These calls can be performed between all Cresco components.

In the next section we demonstrate a Cresco distributed application.

IV. Cresco Demonstrations

The Cresco framework is currently being used for operational data collection, monitoring, and complex event analysis within GENI, distributed clinical genomic processing for a major research hospital, and distributed device control, data collection, and measurement as part of a International Research Network Connections (IRNC) [30] program. Describing the Cresco applications in relation to these programs is beyond the scope of this paper, instead we provide an example demonstrations of Cresco capabilities.

The optimal assignment of workloads between geographic regions and data centers, or edge resources, that account for network communications cost can be addressed as an NP-hard [31], [32] graph partitioning problems. While we make no claims to the advancement of optimization algorithms, we do provide a framework to quickly access global metrics, make new resource assignments, and easily evaluate overall performance impacts.

In our demonstration we will simulate a distributed computational workload between four hypothetical edge or data center regions. Unique data is generated from, and processed, by each region, with fixed aggregate data sources and destinations assigned to each area. The demonstration simulates moving computation to data sources dynamically based on observed performance.

In the example case, workload Actors are implemented in native code as Agent-Plugins. However, the demonstration would work exactly the same if the Agent-Plugins functioned as interfaces to external systems.

Demonstration workloads are described as the following:

- *Workload In*: Every second, a node generates and transmits one universally unique identifiers (UUID) [33] to a queue.
- *Workload Out*: A node receives UUID(s) from a queue and creates a MD5 [34] hash for the associated UUID(s).

Demonstration metrics include:

- *etIn*: The time it takes to generate and transmit a single UUID (32 digit hexadecimal number) to a queue.
- *w*: The time it takes a message spends in the queue.
- *etOut*: The time it takes to transmit a UUID from a queue and create a MD5 Hash (32 digit hexadecimal number) from the UUID.
- *t*: The total time from UUID creation to MD5 creation.

In the demonstration we want to maximize global production by observing workload performance of individual Actors and managing workload assignments within the simulated edge environments.

A. Demonstration Environment

The demonstration, environment is composed of four separate OpenStack [35] tenants. Tenants are groupings of virtually isolated compute, network, and storage resources. Through the use of tenant isolation, we simulate the use of resources from four edge regions. Intra-provider traffic is switched, allowing low-latency communications between hosts on the same tenant. Inter-provider traffic will be routed and then translated

(NATed) on the network increasing communication latency. In the described network configuration, tenants can initiate connections to external destinations, but external destinations can not initiate connection to tenant networks.

The *topology graph* maintains the status of two agents in each of the four regions. In addition, one agent in each tenant acts as the Regional Controller for that tenant. The topology graph is communicated to the global controller (not shown), through regional controllers.

Through the global controller we have access to resources represented in the *topology graph*. To construct the *resource graph*, commands are issued from the global controller to agents to download the workload plugin. Next, the global controller specifies configurations for each agent to be configured with four "in" and four "out" workload plugins. Initially, plugins on each agent will be configured to use a fixed queue in each of the regions.

B. Optimization Procedure

Once a plugin is enabled, performance metrics are reported to regional controllers and propagated to the global controller. The *isConnected* edge represents configuration parameters and performance metrics for a specific plugin, agent, regional, and global application relationship.

From the *resource graph* and associated *isConnected* reports, we can determine the highest performing plugins. Using the *topology graph* we associate plugin performance with plugin, agent, and regional configurations. We both observe performance and take subsequent actions to improve performance through configuration and resource management.

Plugin performance is evaluated in consecutive rounds where the lowest performing plugin configurations in each agent will be replaced with the highest. Performance is evaluated based on minimum workload execution time, measured in nano seconds.

To account for data transfer cost and latency that would typically be experienced in geographically distributed resources, a 20%[3] improvement in reported performance will be added to plugins communicating within the same region.

C. Demonstration Results

The results of the optimization procedure in the demonstration environment is shown in Table I.

Round	*etIn*	*etOut*
0	8.27e7	8.24e6
1	6.78e7	5.68e6
2	5.48e7	4.61e6
3	5.10e7	*4.78e6*
4	4.80e7	4.40e6

TABLE I
MEAN EXECUTION TIME (ET) PERFORMANCE IN MILISECONDS

[3]Percentage based on our enterprise experience of 15-30% per VM for cloud transfer rate cost.

In the demonstration application, mean *etIn* and *etOut* was reduced 42% and 46% respectively. Even with the 20% incentive provided to intra-regional queue assignments, only 46% of plugins were associated with queues found in their own regions. While one would expect interoperating workloads to cluster around regional resources, resource contention within regions can create opportunities for global improvement through intra-regional cooperations. The underlying infrastructure supporting the demonstration environment, much like a public cloud or edge resource, is often a shared service and resource contention will vary from physical server, data center, and even region.

In this simple example, the configuration and reconfiguration of distributed resources to show a global improvement was trivial. However, the way in which resource information was obtained and acted upon was not. We demonstrated how the use of a framework like Cresco could be employed as a cornerstone for more advanced resource evaluations.

V. Related Work

A number of software platforms [36], [37], [38], [39], [40], [41] have been developed to support IoT efforts. IoT platforms provide device management, data transmission encryption [42], support for common data collection protocols [43], and data analysis services. However, the majority of existing platforms rely on public cloud providers such as Amazon EC2 [44] and Microsoft Azure [45] for backend services, requiring the movement of data from sources of collection to remote data centers for processing.

Expanding the research of these and other systems, the goal of the Cresco Framework is to provide a general platform for global application configuration, monitoring, and optimization, through the abstraction of resource instantiations as a graph of metric reporting configurations.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1346688 Subcontract 1939C and ACI-1450937

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VI. Conclusions

In this paper we described our approach to edge computing. We demonstrated a framework for overcoming a described list of logistical challenges. Finally, we described possible approaches to selectively optimize resources on a global-level. Both the presentation of the problem and discussions related to problem solving approaches, are framed in the context of resources found on the edge if a network. However, the demonstrated work and proposed research are applicable to processing in the data center. This is to say, a plugin could easily be any device software/hardware/mechanical

that could be represented by a key-value configuration, and could produce utilization/performance metric(s). For instance, consider an agent integrated into an electric utility meter, where plugins could be assigned for all major electric consumers serviced by that meter for distributed collection, but processing might take place at a regional or cloud data center. Likewise, consider that adaptive scheduling of traffic around a congested city, based on a sensor network deployed under this framework, could make coordinated scheduling decisions without the use of a central coordinator. In each of these cases plugins must be developed and optimization procedures defined, but the core Cresco framework can be reused.

The authors makes no claim to solve a wide range of optimization problems. However, through the development of the non-uniform (variable cost, availability, and performance) resource case specified in this paper, the authors will further demonstrate improvements of globally distributed applications, using edge and cloud computing through advanced market-driven distributed resource optimizations. The outcome of this work will produce a framework that could be adopted for a wide-range of distributed resource management, while demonstrating the framework in the specified non-uniform case presented in this paper.

REFERENCES

[1] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 349–359, Aug 2014.

[2] L. Ericsson, "More than 50 billion connected devices," *Ericsson White Paper*, 2011.

[3] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, pp. 1–11, 2011.

[4] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, pp. 161–166, 2011.

[5] D. Boswarthick, O. Elloumi, and O. Hersent, *M2M communications: a systems approach*. John Wiley & Sons, 2012.

[6] P. C. Evans and M. Annunziata, "Industrial internet: Pushing the boundaries of minds and machines," *General Electric. November*, vol. 26, 2012.

[7] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 22–32, 2014.

[8] 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.

[9] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[10] T. A. S. Foundation, "Apache activemq - mqtt," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/mqtt.html

[11] T. C. Project, "Cresco," 2014, [Online; accessed 11-September-2016]. [Online]. Available: https://github.com/ResearchWorx/Cresco/wiki

[12] T. A. S. Foundation, "Apache license, version 2.0," 2014, [Online; accessed 8-November-2014]. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0.html

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

[14] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger, "The scala language specification," 2004.

[15] M. Thurau, "Akka framework," *University of Lübeck, available online at http://media. itm. uni-luebeck. de/teaching/ws2012/sem-sse/martin-thurauakka. io. pdf [consulted March 29, 2014]*, 2012.

[16] F. Yergeau, "Utf-8, a transformation format of iso 10646," 2003.

[17] V. S. Subrahmanian, *Heterogeneous agent systems*. MIT press, 2000.

[18] S. De Marchi and S. E. Page, "Agent-based models," *Annual Review of Political Science*, vol. 17, pp. 1–20, 2014.

[19] T. A. S. Foundation, "Apache activemq," 2013, [Online; accessed 13-November-2013]. [Online]. Available: http://activemq.apache.org

[20] "Apache activemq - auto," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/auto.html

[21] "Apache activemq - auto," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/amqp.html

[22] "Apache activemq - openwire," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/openwire.html

[23] "Apache activemq - rest," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/rest.html

[24] "Apache activemq - rss and atom," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/rss-and-atom.html

[25] "Apache activemq - stomp," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/stomp.html

[26] "Apache activemq - wsif," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/wsif.html

[27] "Apache activemq - websocket," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/websockets.html

[28] "Apache activemq - xmpp," 2016, [Online; accessed 31-May-2016]. [Online]. Available: http://activemq.apache.org/xmpp.html

[29] O. LTD, "Orientdb," 2016, [Online; accessed 27-May-2016]. [Online]. Available: http://orientdb.com/

[30] K. Thompson and D. Gatchell, "Nsf international research network connections (irnc) program," in *NSF IRNC Program Kickoff Meeting*, 2005.

[31] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[32] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 963–971.

[33] P. J. Leach, M. Mealling, and R. Salz, "A universally unique identifier (uuid) urn namespace," 2005.

[34] R. Rivest, "The md5 message-digest algorithm," 1992.

[35] V. Bumgardner, *Openstack in Action*. Manning Publications Company, 2015. [Online]. Available: http://manning.com/bumgardner

[36] I. Amazon Web Services, "Aws iot," 2016, [Online; accessed 27-May-2016]. [Online]. Available: https://aws.amazon.com/iot/

[37] PTC, "Thingworx," 2016, [Online; accessed 27-May-2016]. [Online]. Available: http://www.thingworx.com/

[38] LogMeIn, "Xively," 2016, [Online; accessed 27-May-2016]. [Online]. Available: https://xively.com

[39] B. I. Suite, "Aws iot," 2016, [Online; accessed 27-May-2016]. [Online]. Available: https://www.bosch-si.com/products/bosch-iot-suite/platform-as-service/paas.html

[40] SensorCloud, "Sensorcloud," 2016, [Online; accessed 27-May-2016]. [Online]. Available: http://sensorcloud.com

[41] IoTivity, "Sensorcloud," 2016, [Online; accessed 27-May-2016]. [Online]. Available: http://sensorcloud.com

[42] J. Granjal, E. Monteiro, and J. Sa Silva, "Security for the internet of things: a survey of existing protocols and open research issues," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 3, pp. 1294–1312, 2015.

[43] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 4, pp. 2347–2376, 2015.

[44] E. Amazon, "Amazon elastic compute cloud (amazon ec2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[45] M. Inc., "Microsoft azure," 2014, [Online; accessed 8-December-2014]. [Online]. Available: http://www.azure.com