An Edge-Focused Model for Distributed Streaming Data Applications

V. K. Cody Bumgardner
Department of Pathology and Laboratory Medicine
University of Kentucky
Lexington, Kentucky, USA
Email: cody@uky.edu

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

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

***Abstract*—This paper presents techniques for the description and management of distributed streaming data applications. The proposed solution provides an abstract description language and operational graph model for supporting collections of autonomous components functioning as distributed systems. Using our approach, applications supporting millions of devices can be easily modeled using abstract configurations. Through our models applications can implemented and managed through so-called server-less application orchestration.**

I. Introduction

The number of connected data-generating devices is rapidly increasing. According to estimates, over 5000 new devices will be added per minute between 2017 and 2018, with overall connected device counts in the tens of billions by 2020 [1]. Connected devices include low-power sensors with consistent data generation rates, and large sporatic data generators, such as the Large Hadron Collider (LHC) [2]. Unlike computational simulations in high-performance clusters and computing clouds, where often information remains in close proximity to computational resources and results are disseminated, connected devices share the raw data they generate, often transmitting data over long distances to remote destinations. Connected devices are not just data producers, they also consume data, which could come from geographically distributed sources. It's common for every subsequent generation of network technology to deliver an order of magnitude improvement in bandwidth capacity [3] and public clouds provide seemingly endless computational resources. However, as demonstrated through Fog [4] and Edge [5] computing efforts, faster communications and remote clouds are not enough to harness the deluge of often transient information generated by an ever increasing number of sources. The coordinated collection, analysis, and transmission of data generated by dispersed and often disparate devices is long-standing area of distributed systems research. The authors believe that if properly described, a number of distributed computational techniques can be brought to bear when designing end-to-end streaming data applications.

We propose the Cresco Application Model (CAM), a graph-based modeling language describing functional units, unit relationships, and operating characteristics used in the deployment and management of distributed streaming data applications. In the following sections we present the purpose of our work, our proposed model, provide analysis of our approach, and report experimental results.

II. System Model

On a fundamental level streaming data applications can be decomposed into data source, intermediate processors, and data drain component types. As the name suggest, data sources generate or feed data into the system, intermediate processors manipulate data, and data drains function as gateways to other applications or long-term storage systems. We assume a one-to-many relationship between source inputs and outputs, many-to-many relationship for intermediate processors, and a many-to-one relationship for data drains.

The authors believe that the following characteristics and operating parameters are essential in the management of distributed streaming applications:

- *Dynamic provisioning*: The underlying infrastructure of large distributed systems composed of heterogeneous components will always be in a state of change. In addition, the data-driven resource demands of applications are also prone to change. Applications must be able to provision and re-provision functional units in response to observations, predictions, or prescribed landscape changes. In this context, a functional unit can be thought of as critical code and software dependencies required to execute one or more well-defined tasks.
- *Autonomous operations*: Once provisioned, functional units should operate autonomously and tolerate intermittent control and data channel interruptions. For example, if a unit loses contact with its control system, it should continue to operate based on current configuration, while attempting to re-establish administrative communications. Likewise, if a functional unit detects a data plane failure while communicating with a downstream node, it should preserve (to the extent local resources allow), data to be transmitted once communication has been re-established.
- *Component- and system-level health and performance evaluations*: Functional units should periodically self-report health and user-defined key performance indicators (KPI) metrics for administrative purposes. In addition, units should take action to avoid or correct predicted

978-1-5386-3227-7/18/$31.00 ©2018 IEEE

and observed component-level breaches of KPI metrics. Administrative-system-supporting collections of components should act on predicted and node-reported problems through the manipulation of new unit configurations, resources levels, and functional unit assignments.

- *Modular scalability*: The configuration of functional units, especially those using intermediate processors, should be replicable, allowing for linear scaling of associated workloads. In addition, instances of functional units should be able to either directly, or through administrative request, self-replicate as determined through KPI analysis.
- *Fault tolerance*: As described in the previous point, functional units report health status periodically allowing administrative systems to detect both known and unknown failures. Known failures are reported directly by units and unknown failures are determined by the absence of so-called *watchdog* notifications within a predetermined interval. With the exception of units with unique location or hardware assignments, such as a specific physical sensor gateway, most functional unit configurations should be re-provisioned on the event of failure. In addition, dynamic provisioning and autonomous operations principles allow for the potential re-provisioning of one or more components without the loss of data.
- *System Optimization*: In cases where KPI and other observable metrics are available, collections of functional units supporting applications should be continually evaluated based on predefined optimization criteria. When administrative systems determine that improvements are possible, reassignments should be made if dynamic provisioning and autonomous operations principles can guarantee no loss of data or breach of KPI.

III. Cresco Application Model (CAM)

A number of templates and related orchestration systems have been proposed for infrastructure [6], platform [7], and multi-cloud [8] application deployment. We make no claims to improve upon existing cloud orchestration platforms. In fact, a CAM streaming data application might make use of underlying infrastructure or software dependencies provided by one or more cloud orchestration tools. We focus on addressing declarative relationships between data flow and functional units operating within intent-based parameters. This is to say, we assume an existing application-layer abstraction with limited control or insight into underlying communications networks, clouds, or associated systems.

In this section we cover the format of the Cresco Application Description Language (CADL), an external descriptive model, and we cover the Cresco Graph Database (CGD), the internal operational model representing distributed streaming data applications.

A. Cresco Application Description Language (CADL)

We find it convenient to model the descriptions of distributed applications and their related data flows as directed graphs, as defined below:

- *Nodes*: Graph nodes are descriptions of context-based functional units defined by abstract configurations. Nodes provide one or more configurable data input and output specifications. In addition, nodes can optionally provide specific KPI for component evaluations. For example, a streaming complex event processing (CEP) [9] node might describe a self-contained system to retrieve data from a queue, perform some action on incoming data, and emit data to a queue when specific conditions are met.
- *Edges*: Graph edges define the relationships between node instances and the potential data transformation between node instances. For example, an edge relating two CEP nodes might convert the output of an upstream node to a format that is usable by a downstream node.
- *Pipelines*: Pipeline (graphs) are collections of nodes and edges that abstractly describe the arrangement, function, and performance requirements of distributed applications. For example, a pipeline application might define a hierarchy of upstream and downstream CEP nodes that allow for high-rate distributed data processing with low-rate aggregated result reporting.

The format of CADL node fields and requirements are shown below:

- **[node_id]** *(required, unique)*: Node IDs are unique identifiers for nodes within specific pipelines.
- **[node_name]** *(required)*: Node names are used as short descriptions for nodes within specific pipelines.
- **[type]** *(required)*: Node types represent specific functional unit implementations.
- **[description]** *(optional)*: Node descriptions are used for descriptions of node operations within specific pipelines.
- **[params]** *(optional)*: Params are collection of key-value pairs that specify functional unit configurations.
- **[kpis]** *(optional)*: KPIs are collection of key-value pairs that specify KPI operating parameters.
- **[isStateless]** *(optional)*: The isStateless parameter is a boolean value representing the ability of the configuration instantiation to be migrated without maintaining the current state.
- **[isUnique]** *(optional)*: The isUnique parameter is a boolean value designating whether the node's assigned location and configuration must remained unchanged.
- **[location]** *(optional)*: The location parameter is used to relate nodes to specific agents or locations.

Listing 1 shows a node description for a stateless CEP function unit with no designated location.

```
"node_id":"0"
"node_name":"pStart"
"type":"CEP"
"params":
  "input_class":"flow"
  "output_class":"flow"
  "query":"select * from flow where x > 0"
"isStateless":true
```

```
"isUnique": false
"location":""
```

Listing 1. CADL Node

As previously mentioned, edges represent the relationship between nodes. The format of CADL edge fields and requirements is shown below:

- **[edge_id]** *(required, unique)*: Edge IDs are unique identifiers for nodes within specific pipelines.
- **[node_from]** *(required)*: The node_from parameter is used to designate source node_id for the edge within a specific pipeline.
- **[node_to]** *(required)*: The node_to parameter is used to designate destination node_id for the edge within a specific pipeline.
- **[input_type]** *(optional)*: Type of required upstream data source.
- **[output_type]** *(optional)*: Type of required downstream data destination.

Listing 2 shows an example of a CADL edge description relating two node_ids.

```
"edge_id":0,
"node_to":"1",
"node_from":"0"
"input_type":"AMPQ"
"output_type":"AMPQ"
```

Listing 2. CADL Edge

CADL node and edge descriptions are combined to form a pipeline description, which represents a collection of managed components forming an application. The format of CADL pipeline fields and requirements are shown below:

- **[pipeline_name]** *(required)*: Pipeline names are used as short descriptions for pipelines maintained by a specific Cresco Global Controller.
- **[nodes]** *(required)*: Nodes are collections of CADL node descriptions. At least one node description must exist for a pipeline to be considered valid.
- **[edges]** *(optional)*: Edges are collections of CADL edge descriptions.
- **[description]** *(optional)*: Pipeline descriptions are used to describe the operation of pipelines.
- **[isFaultTolerant]** *(optional)*: The isFaultTolerant parameter is a boolean value designating if pipeline components should be rescheduled if failures are detected.

B. Cresco Graph Database (CGDB)

As with other template languages, the CADL provides enough information for an application management framework to statically provision a component pipeline. However, to implement the desired application management characteristics described in Section II, *System Model*, we need a model with the ability to describe more than just abstract component descriptions. The CGDB is used to model abstract (CADL virtual nodes), operational (node instantiations), and data translations (nodes generated in edge provisioning).

The following node and edge class instances are used to describe pipelines in the CGDB model.

- **Node** *pipelineNode* : pipelineNodes are root nodes for pipelines described by the CADL language.
- **Node** *vNode* : vNodes maintain a record of pipeline nodes as described by the CADL language.
- **Node** *iNode* : iNodes represent instantiation resource assignments related to vNodes.
- **Node** *eNode* : eNodes represent data exchange mechanism between iNodes. eNodes maintain configuration information such as node configurations generated to fulfill data exchange between differing node output and intput types.
- **Edge** *isVconnected* : isVConnected associates vNodes that are connected based on CADL description.
- **Edge** *isEconnected* : isEconnected edge indicates the directed flow of data from one eNode to another.

The relationship of CGDB classes is shown in Figure 1.

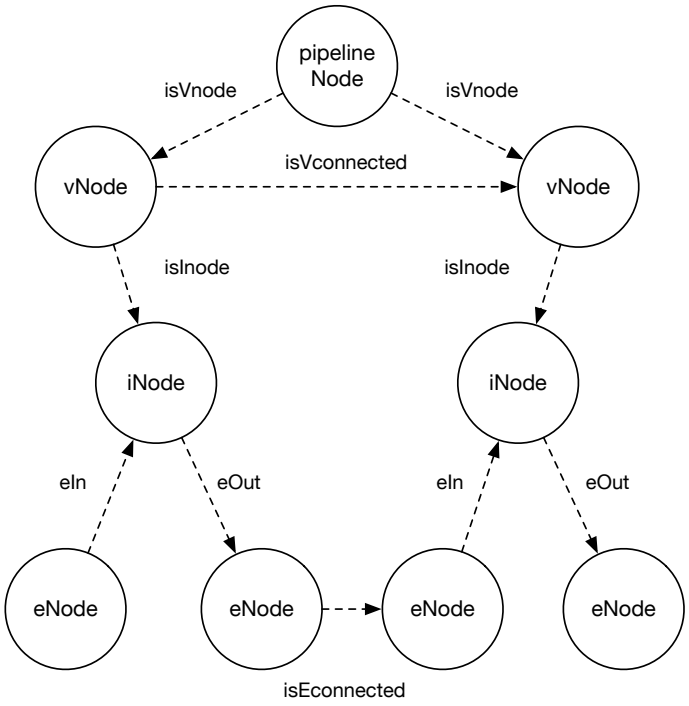

Fig. 1. Cresco Application Graph

CADL nodes are represented as vNodes in the database. There is a one-to-many relationship between iNodes and vNodes, allowing iNode to be shared between pipelines. If needed, iNodes are created for each vNode to represent implementations of requested vNodes. On iNode, and related eNode generation, CGDB models are searched for existing candidate iNodes with the exact same configuration parameters and data path(s) as the iNode representation of pipeline-specific vNodes. If an iNode replacement candidate is found, "isEConnected" and "eIn" relationships can be traversed to determine if iNode and eNode implementations for the source data path are exactly the same as the sub-graph to be implemented.

If a valid iNode sub-graph is found, the vNode "isINode" relationship is assigned to the existing iNode. The avoidance of new iNode creation serves as a system-wide data and resource de-duplication [10]. Implementation nodes (iNode and eNode) along with edges, are updated with observed KPI and, possibly,other metrics. The CGDB model provides information to higher-order management systems used in the maintenance of large pervasive streaming data applications.

IV. Analysis

The previous section covered a text-based language to describe, and a graph structure to operate streaming data applications. In this section we analyze how the models are used to support a subset of characteristics and operating parameters important in distributed streaming data applications, as described in Section II, *System Model*.

A. Dynamic Provisioning

Software-defined provisioning and distributed resource management is an active area of research. Geographic Load-balancing (GLB) [11] scheduling systems such as Nimrod/G [12] for high performance computing and DONAR [13] for content replication, exist. We are not concerned with the low-level provisioning of infrastructure, we focus on the appropriate assignment of modular workload based on observed workload resource requirements, node-defined KPIs, and measured supporting environmental changes.

We define dynamic provisioning as the arrangement (initial and repeated) of provisioned resources from available resource providers. Based on operational policy (e.g. workload data can not leave specified site) and technical best practices (e.g. separation of latency-dependent resources across high-latency connection), not all available resources will be candidates for assignment. This is to say, if we want to assign resources, we need to decide where in a distributed environment can and should resources be assigned.

Given the set S of observable resources, we assume that there exist a subset T of resources that satisfy assignment requirements as described by a node configuration ($T \subseteq S$). If a resource is contained in the set T, it is considered available for assignment. The available set is constructed by eliminating resources from the global set. Resources that violate Service Level Agreement (SLA) [14] and usage policy constraints [15] described by configuration can be directly eliminated from the global set.

To provide some intuition, let us assume that we are building an application requiring a certain number of queue resources, $R_0, \ldots R_{m-1}$ with available bandwidth $a_0, \ldots, a_{m-1}$, respectively.

Next, we assume that we have at our disposal k queue systems, $D_0, \ldots, D_{k-1}$ with the bandwidth capabilities, $b_0, \ldots, b_{k-1}$, respectively. The space for resource R_i must be assigned at one of D_i's. With such constraints, we can see that the desired assignment reduces to solving a system of inequalities:

$$\begin{aligned} a_0 x_{0,0} + \quad & a_1 x_{0,1} + \ldots + a_{m-1} x_{0.m-1} \leq b_0 \\ & \ldots \\ a_0 x_{k-1,0} + \quad & a_1 x_{0,k-1} + \ldots + a_{m-1} x_{k-1,m-1} \leq b_{k-1} \end{aligned}$$

subject to the number of constraints. The basic constraints are:

1) The solution is *pseudo-Boolean*, i.e. variables $x_{g,h}$ take only values 0 and 1.
2) For each $j, 0 \leq j \leq m-1$ exactly one of $x_{j,0}, \ldots x_{j,k-1}$ takes value 1.

We observe that the proposed model is quite flexible since we can impose additional constraints, for instance requiring that some resources are placed on a specific D_i (or within specific group of D_i's). Likewise negative constraints (R_j not to be placed within a group of D_is) can also be easily expressed.

The restriction that we are dealing with pseudo-Boolean solutions allows the use of tools such as reduced ordered binary decision diagram (ROBDD) for fast checking the satisfaction of constraints [**?**]. We also observe that there is a large body of knowledge [16] on solving systems of inequalities over integers (including a three-volume classical book by A. Schrijver. [17]).

The above model dealt with placement of a single resource (we illustrated the approach with queue resources assignment). In practice, however, more than one resource may be considered.

B. Modular scalability

The study of workload arrival, servicing, and departure is referred to as *queueing theory* [18]. In theory, the duplication of a critical function unit would double the potential aggregate output of that system. Likewise, the deduplication of a functional unit would provide the same output at a reduced resource cost. In the CAM, iNodes can both be replicated within pipelines to provide increased performance and deduplicated across pipelines to reduce resource needs.

C. Fault tolerance

Through CADL location assignments, application nodes can be geographically distributed for the purposes of proactive fault tolerance. Through the use of CGDB modeling, we know the abstract relationship between the intent-described vNode configurations and system-derived iNode implementations. In the event an iNode fails, a new iNode can be generated from the vNode and one or more pipelines can be reconfigured to use the new iNode.

D. System Optimization

In Section IV-A, *Dynamic Provisioning*, we described a *pseudo-Boolean* approach to dynamic node provisioning. In this section, we describe a greedy [19] problem solving heuristic used determine locally and potentially global optimal workload assignments.

This algorithm will be explained in the context of CADL, CGDB, and based on the five general components of greedy algorithms:

1) *Candidate Set*: Candidate sets are composed of node configurations or instantiations that make up a pipeline application. These sets might contain one or more explicit resource requirement and/or observed workload utilization related to KPI metrics.
2) *Selection Function*: It is useful to think of computational resources in terms of commodity resources (water, electricity, gas, etc.), like those that are traded in commodity markets [20]. In this context, a commodity is a resource that is considered the same regardless of the originating provider. While this seems to contradict previous statements related to the variability of resources across providers, we will account for this variation in objective function.
3) *Feasibility Function*: The process of determining feasibility was previously described in In Section IV-A, *Dynamic Provisioning*.
4) *Objective Function*: The objective function is dependent on the type of optimization. An objective function must evaluate a number of factors in order to assign value to a solution. In the commodity trade, actors (providers and consumers) agree on contracts to describe the fulfillment of a commodity transaction. Contracts are made up of the following four components, which we will relate to computational resources:
   - *Quantity*: the resource count related to the atomic *horizontal scaling* of an application component.
   - *Quality*: The selection function evaluates the performance of resource providing services through metrics determined through active and simulated methods. In the active case, self-monitoring [21] data comparing resource utilization with workload KPI metrics is used directly. However, if such data is not available, synthetic micro-benchmarks [22] can be generated for potential resource providers.
   - *Price*: is the actual billed price of the resource, typically a product of time and reserved capacity. Price could be set by a market (Amazon EC2 [23], Azure [24], etc.) or they could simply be the fixed cost assigned to a local resource. Cost accounting models [25], [26] exist to provide market comparisons between resource options. In addition, one must account for *Cost of Risk*, which is the cost allocated to the potential of resource compromise. In this context, compromise is defined as a loss of data integrity, data exposure to unauthorized parties, or high variability in resource performance. Risk models [27], [28] have been developed to provide cost-of-risk estimations.
   - *Delivery*: can be considered the cost of converting or migrating [29] potential resources (configuration, data, virtual machine, etc.), to active resource. This cost is typically the sum of data transfer cost and deployment (startup, configuration, transfer time) cost.
5) *Solution Function*: A globally optimal solution can not be verified using this method. However, a global improvement related to a defined workload optimization can be verified.

We claim that the overall solution can be expressed and evaluated using the described methods in the context of resource contracts. While we have described the process for evaluating a single application, existing assigned resources known to a higher-level administrative system should also be taken into account. A natural fragmentation of distributed applications and resources can occur if the evaluation and reassignment of active nodes does not take place. One possible solution is to evaluate node configurations and their related commodity resource contracts for resource reassignment. Similar commodity resource contracts can be used in market equilibrium (balance supply and demand) [30] resource scheduling allowing the evaluation of assigned resources as if they are available.

V. Simulation and Experimentation

Smart Cities applications such as traffic and environmental sensor management require data processing on street-intersection, neighborhood, and city-wide levels. Potential data sources include distributed sensor arrays, vehicles, and personal devices. Distributed resources might be used for data interoperation, processing (analysis) services, and the coordination of information, such as autonomous Vehicle-to-Vehicle (V2V) interactions. The coordination of millions of potential devices and sources of data in a large metropolitan area is a serious computational challenge.

We have developed data source (street-intersection), intermediate processor (neighborhood), and data drain (central monitoring system) functional units to model and simulate the devices, flow of data, and operation of 1 million devices across a large smart city. We assume that each street intersection has at least 1000 reportable devices (data points). We assign 100 street-intersection nodes to each neighborhood and 100 neighborhoods to a central monitoring system. The CADL representation of this pipeline contains a graph with 30,000 street-intersection nodes and 40,000 edges, 500 neighborhood nodes and 700 edges, and three central monitor nodes and two edges. The described graph results in a CADL size of 27.4 (1.3 compressed) Megabytes.

Using the CGDB model, we provisioned our application across 20 nodes, each with 8 cpu cores, 8G of RAM, and a 128G disk. We assume street-level operation takes place in a location with limited computational capacity, such as an intelligent street light. On a street-level, we want to simulate the collection of data from sensors, which should be physically connected or in proximity of collection devices and are not candidates for reassignment. We generate and transmit 1000 fixed (sensors data) and 10-70 variable (vehicle data) values for each node per second. When an alert is triggered, a notification is sent to the neighborhood. We assume neighborhood-level operations take place in small-to-medium sized telecommunication facilities distributed throughout a city, with enough

computational capacity to provide analytic services for a network of associated gateways. Locations were not assigned to Neighborhood nodes. Neighborhood nodes see between 151,500 and 160,500 data points per second. Each event is observed by one or more CEP engines, which can trigger upstream alerts and downstream changes in data reporting. In response to KPI metrics and related to values generated on the street-level, neighborhood-level nodes can expand and contract instances of their configuration as needed. Overall, the central monitoring system in this simulation processes between 2,878,500 and 3,049,500 metrics per second, and like neighborhood nodes can expand, contract, and recover from fault as needed.

VI. Conclusion

This paper presented techniques used to describe, model, and analyze distributed streaming data applications. We have shown how the presented data model can be used in dynamic provisioning, modular scalability, fault tolerance, and systems optimization. In addition, through experimentation we have shown that our methods can be used to model large (1 million devices) distributed applications. Along with simulation, the described methods have been implemented in part by the Cresco Framework [31] and are used by a number of real-world applications. We will continue to improve this work as the overall framework advances.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 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.

References

[1] "Gartner says 8.4 billion connected ?things? will be in use in 2017, up 31 percent from 2016," Feb 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[2] G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz, E. Acerbi *et al.*, "The atlas experiment at the cern large hadron collider," *Journal of Instrumentation*, vol. 3, no. 8, pp. S08 003–S08 003, 2008.

[3] O. Fagbohun, "Comparative studies on 3g, 4g and 5g wireless technology," *IOSR Journal of Electronics and Communication Engineering*, vol. 9, no. 3, pp. 88–94, 2014.

[4] 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.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] T. Faber and R. Ricci, "Resource description in geni: Rspec model," in *Presentation given at the Second GENI Engineering Conference (March 2008)*, 2008.

[7] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: Portable automated deployment and management of cloud applications," in *Advanced Web Services*. Springer, 2014, pp. 527–549.

[8] K. Alexander, C. Lee, E. Kim, and S. Helal, "Enabling end-to-end orchestration of multi-cloud applications," *IEEE Access*, vol. 5, pp. 18 862–18 875, 2017.

[9] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 407–418.

[10] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality." in *Fast*, vol. 9, 2009, pp. 111–123.

[11] J. Camacho, Y. Zhang, M. Chen, and D. Chiu, "Balance your bids before your bits: The economics of geographic load-balancing," *Proc. of ACM e-Energy*, 2014.

[12] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," in *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, vol. 1. IEEE, 2000, pp. 283–289.

[13] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 231–242, 2010.

[14] L. Wu, S. K. Garg, R. Buyya, C. Chen, and S. Versteeg, "Automated sla negotiation framework for cloud computing," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 235–244.

[15] A. M. Oprea, Y. Zhang, V. Ganti, J. P. Field, A. Juels, and M. K. Reiter, "Security policy enforcement framework for cloud-based information processing systems," Apr. 1 2014, uS Patent 8,689,282.

[16] B. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2002.

[17] F. Somenzi, CUDD page, http://vlsi.colorado.edu/ fabio. Accessed December 2017.

[18] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

[19] L. Kleinrock, *Queueing systems, volume 2: Computer applications*. wiley New York, 1976, vol. 66.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.

[21] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[22] J. S. Ward and A. Barker, "Self managing monitoring for highly elastic large scale cloud deployments," in *Proceedings of the sixth international workshop on Data intensive distributed computing*. ACM, 2014, pp. 3–10.

[23] T. Li, I. Raicu, and L. Ramakrishnan, "Scalable state management for scientific applications in the cloud," *IEEE BigData*, 2014.

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

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

[26] I. Ruiz-Agundez, Y. K. Penya, and P. G. Bringas, "A flexible accounting model for cloud computing," in *SRII Global Conference (SRII), 2011 Annual*. IEEE, 2011, pp. 277–284.

[27] E. Elmroth, F. G. Marquez, D. Henriksson, and D. P. Ferrera, "Accounting and billing for federated cloud infrastructures," in *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*. IEEE, 2009, pp. 268–275.

[28] B. Martens and F. Teuteberg, "Decision-making in cloud computing environments: A cost and risk based approach," *Information Systems Frontiers*, vol. 14, no. 4, pp. 871–893, 2012.

[29] S. Paquette, P. T. Jaeger, and S. C. Wilson, "Identifying the security risks associated with governmental use of cloud computing," *Government Information Quarterly*, vol. 27, no. 3, pp. 245–253, 2010.

[30] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Cloud Computing*. Springer, 2009, pp. 254–265.

[31] R. Arfa and J. Broeckhove, "Modeling resource prices in computational commodity markets," in *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*. IEEE, 2009, pp. 145–152.

[32] V. C. Bumgardner, V. W. Marek, and C. D. Hickey, "Cresco: A distributed agent-based edge computing framework," in *Network and Service Management (CNSM), 2016 12th International Conference on*. IEEE, 2016, pp. 400–405.