A Resilient Software Infrastructure for Wide-Area Measurement Systems

Tao Qian[1], Hang Xu[1], Jianhua Zhang[2], Aranya Chakrabortty[2], Frank Mueller[1]
[1]Department of Computer Science, [2]Electrical Engineering
North Carolina State University
Raleigh, NC, USA
{tqian2,jzhang25,aranya.chakrabortty,fmuelle}@ncsu.edu

Yufeng Xin
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
Chapel Hill, NC, USA
yxin@renci.org

Abstract—**To support the scalability and resilience requirements of distributed Wide-Area Measurement System (WAMS) architectures, we design and implement a software infrastructure to estimate power grid oscillation modes based on real-time data collected from Phasor Measurement Units (PMUs). This estimation algorithm can be deployed on a hierarchical structure of Phasor Data Concentrators (PDCs), which calculate local estimates and communicate with each other to calculate the global estimate. This work contributes a resilient system to WAMS with guarantees for (1) Quality of Service in network delay, (2) network failure tolerance, and (3) self-recoverability. The core component of the infrastructure is a distributed storage system. Externally, the storage system provides a cloud data lookup service with bounded response times and resilience, which decouples the data communication between PMUs, PDCs, and power-grid monitor/control applications. Internally, the storage system organizes PDCs as storage nodes and employs a real-time task scheduler to order data lookup requests so that urgent requests can be served earlier. To demonstrate the resilience of our distributed system, we deploy the system on a (1) virtual platform and (2) bare-metal machines, where we run a distributed algorithm on the basis of the Prony algorithm and the Alternating Directions Method of Multipliers (ADMM) to estimate the electro-mechanical oscillation modes. We inject different failures into the system to study their impact on the estimation algorithm. Our experiments show that temporary failures of a PDC or a network link do not affect the estimation result since the historical PMU data are cached in the storage system and PDCs can obtain the data on demand.**

I. Introduction

With the continuing large-scale deployment of PMUs, the current centralized power grid monitor and control infrastructure of the WAMS technology is beginning to show limitations [1]. First, the computation limitations of a centralized server may result in a lag in wide-area monitoring that falls short of real-time requirements. Second, network contention could become severe making it hard to satisfy QoS requirements of monitor and control applications since a significant number of PMUs are sending data to the centralized server simultaneously. In response, WAMS technology is starting to embrace a distributed architecture where multiple sets of PDCs process PMU data to support power grid intelligence with real-time constraints in a distributed manner. In recent work [2], we propose a cloud computing based virtual smart grid (vSG) framework that supports the dynamic creation of distributed applications in clouds to connect to a set of target PMUs and to store/process their sensor data in real-time. This architecture distributes computation among multiple PDCs, which addresses the aforementioned disadvantages of a centralized architecture.

Another objective is to provide resilience to compute or communication failures for distributed power grid monitoring. PMUs have to maintain a communication channel with their corresponding PDC to send monitored data to the PDC. In case of a communication channel failure or PDC failure, only the most recent monitored data is lost. In addition, without regulations on how data are processed and communicated among different data sources (e.g., PDCs, WAMS applications), distributed algorithms must by design handle unpredictable network delays from different data sources.

We have addressed these problems in a distributed data storage middleware that is placed in the midst of PMUs, PDCs and WAMS applications [3], [4]. Our distributed data storage system provides two API services: *put(key, data)* stores data with a *key* as its index; *get(key)* returns the data associated with a *key*. The system supports a storage abstractions where PDC nodes create a distributed hash table (DHT). DHTs, such as Chord [5] and CAN [6], are well suited for this problem due to their superior scalability, reliability, and performance characteristics compared to centralized or tree-based hierarchical approaches.

Our distributed system focuses on providing resilience to WAMS in three aspects. First, considering the real-time requirements of WAMS, the network delay of communication between PDCs must be upper bounded even if the network is congested. To fulfill the **QoS** requirements of network delay, we employ a real-time task scheduler on storage nodes to serve data lookup requests with bounded end-to-end response times. Second, occasional network link failures between PDCs must not affect the WAMS algorithm. Without our distributed system, PDCs communicate with each other via direct network channels. This communication seizes to work when a single network link along the channel fails. In contrast, our distributed system consists of multiple autonomous storage nodes to decouple the tight connection between PDCs. In detail, PDCs store their data in our distributed storage and fetch new data from this storage system on demand. Since these storage nodes have multiple access points, a network failure between a PDC and a storage node does not affect the overall data availability or consistency. This PDC can utilize an alternative storage node to access data (**network fault tolerance**). Third, our storage layer employs a DHT algorithm to implement a distributed storage that replicates data across its nodes. As a result, the storage system can serve requests even after node failures without data loss (**self-recoverability**).

We implement an earliest-deadline-first (EDF) task scheduler and an EDF packet scheduler to prioritize requests by their deadlines within the real-time distributed storage abstraction. The EDF scheduler reduces the time required for failure recovery, since an application can increase the priority of requests by shortening deadlines in recovery mode (after a node/link failure).

We implement our distributed storage system on Linux and deploy it both on bare-metal machines and a virtual platform (ExoGENI [7]). In experiments, we deploy a distributed algorithm based on Alternating Direction Multiplier Method (ADMM) for estimating electro-mechanical oscillation modes [1] to demonstrate how the storage system is utilized and how the design of distributed algorithms is facilitated by our storage abstraction. In addition, we simulate different failures and demonstrate the resilience of our system, which current centralized wide-area monitoring and control infrastructure cannot provide.

II. A Resilient Distributed Infrastructure

A. The Centralized Measurement System

Fig. 1 depicts the centralized measurement system infrastructure. A PMU transmits monitoring data periodically to its connected local PDC. In one iteration of the ADMM algorithm, the local PDCs run the estimation algorithm based on their PMU data and send the local estimates to the central PDC. Then, the central PDC estimates the global state based on the data received from local PDCs and transmits its result back to the local PDCs. Subsequently, the local PDCs start a new iteration of the estimation algorithm based on the global estimation and new PMU data. These iterations continue until the estimation on all PDCs converge. Section III details this algorithm. The architecture involves multiple PDCs organized in a client/server style, where a central PDC coordinates actions in a distributed envirnment, much in contrast to fully distributed algorithms without any single point of failure. For the rest of the paper, we will refer to the architecture simply as *centralized*.

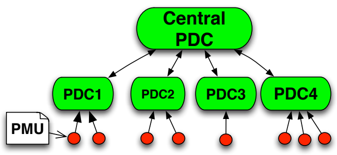

Fig. 1. Centralized Measurement System Example

The general ADMM algorithm has been well studied. However, several details have to be considered in the implementation for a real estimation system. First, the stability of the network channel between the local PDCs and the central PDC has a significant impact on estimation accuracy. If the network channel between PDC1 and the central PDC has failed, the data from the corresponding PMUs are unknown to other PDCs. Second, the central PDC has to consider the transmission time on each link to determine when to estimate the global state in the current iteration. When some PDC data has not arrived within a time threshold, it is difficult for the central PDC to determine whether the long delay is due to a failed PDC, a network failure, or temporary network congestion. We address these challenges in our resilient distributed infrastructure for WAMS in the next section.

B. A Resilient Real-time Storage System

Our real-time storage system is the core component of our resilient wide-area measurement systems. As shown in Fig. 2, the storage system acts as a middleware between local PDCs and the central PDC to cache the estimation data.

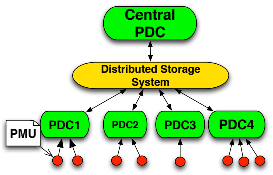

Fig. 2. Measurement System with Real-time Storage System

Externally, our storage system provides two fundamental cloud services: *get* and *put*. PDCs and WAMS applications utilize the *put* service to store monitoring data or intermediate power grid estimation results. They can further use the *get* service to obtain relevant data from the storage system. Thus, this distributed storage system provides an additional protocol layer between data providers (PMUs/PDCs) and data consumers (PDCs and WAMS applications).

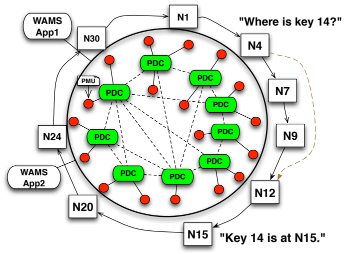

Fig. 3. PDC, PMU, and Chord Ring Mapping Example

Internally, we utilize a distributed hash table (DHT) over a set of storage nodes (chosen to also be the PDCs) so that the power grid data can be disseminated in a distributed manner and subsequently accessed by monitoring and control applications. Fig. 3 illustrates how the DHT protocol organizes PDCs as storage nodes in a virtual Chord-like ring structure [5]. This ring structure provides a natural way to orchestrate power estimates and, optionally, actuation tasks of disjoint PDCs based on key/value pairs. The storage system can store raw PMU data, memorize state and parameter estimates, and even actuation intentions. In Fig. 3, PMUs, PDCs, and WAMS applications logically access our distributed storage system and coordinate actions with one another. In this example, 9 PDCs are mapped onto the Chord ring and utilized as DHT storage nodes. A PMU sends raw monitoring data to the storage system via a connection with its local PDC. However, the raw PMU data are not necessarily stored in the local PDC. Our storage system utilizes a consistent hashing algorithm [8] to map data to virtual nodes. For example, the data with key 14 is located on virtual node 15 according to the Chord algorithm. The WAMS application may send requests to any PDC in the ring to fetch the data on demand. It is sufficient to locate any data by maintaining the nodes in a wrap-around circular list such that each node has a reference to its successor node, i.e., a ring traversal can always locate the data. However, this linear search algorithm is not scalable with increasing numbers of PDCs. Chord utilizes finger tables to reduce the number of intermediate nodes to $log(N)$ level, where N denotes the number of DHT nodes (see our previous work [3] for details).

This storage system improves the stability of the overall system in three ways. First, since the data are disseminated in a distributed manner on these storage nodes, any network link failure between a PDC and a storage node will not result in the data loss that a centralized measurement system experiences. Instead, the PDC chooses an alternative node to put/get data. We further adopt a real-time task scheduler on storage nodes so that the response time of data transmission is predictable. Third, since these storage nodes run autonomously and create replicas of data in different storage nodes, a single storage node failure will not result in data loss.

Data security needs to be considered in the storage system since the data are disseminated among different storage nodes. Since the *put* and *get* provided by our storage system are general cloud services without any requirement for the format of the data content, PDCs can easily integrate a public-key cryptography with the storage system to secure data. For example, PDCs can utilize RSA, an asymmetric encryption algorithm, to encode data before transmitting it to storage nodes. Then only the central PDC, which has the corresponding private key, can decode the data.

One important part of our distributed storage system is to provide QoS for the response time of any data access. To this end, we adopt a hybrid EDF scheduler so that urgent data requests are served at higher priority, i.e., ahead of lower priority requests that were issued earlier but have not yet been processed. This hybrid EDF scheduler includes two components: the EDF task scheduler, which schedules data requests in EDF order, and the EDF packet scheduler, which transmits IP packets that carry data request messages in EDF order. As a result, the deadlines carried in data messages are considered at the application layer as well as the network layer of the storage abstraction.

We have extended the Linux kernel to implement this hybrid EDF scheduler. Since the Linux traffic control layer does not support task deadlines embedded in data messages, which are encapsulated by the application layer, we extended the data structure for IP packets in the Linux network stack by adding new fields to store timestamps. We also extended the *setsockopt* system call so that it supplies these timestamps upon request. The most significant changes to Linux to support this functionality are as follows:

(1) We added a new field in the kernel data structure to store the deadline of a socket transmission. (2) We extended the kernel function *sock_setsockopt* with option $SO\_DEADLINE$, so that applications can specify message deadlines associated with messages via $setsockopt$ in user mode. (3) When the application transmits a message, the kernel stores the message data including the *deadline* of the socket. After this, the deadline of the message is passed down to the transport layer. (4) We implemented an EDF packet scheduler, which provides the standard interface of Linux traffic control queue disciplines [9]. The EDF packet scheduler utilizes a prioritized queue to maintain messages in a min-heap data structure as a linked list.

These novel extensions provide the capability of specifying message deadlines for real-time tasks (applications). With these provision, our EDF packet scheduler utilizes the message deadlines to transmit packets in EDF order.

III. WAMS on Real-time distributed storage

We next show how the electro-mechanical mode estimation algorithm proposed in our recent paper [1] can be integrated with real-time storage to deposit/retrieve data and communicate between PDCs.

A. Prony and ADMM Algorithms

The problem of estimating the electro-mechanical oscillation modes can be cast in discrete-time domain as least-squares estimation of the common characteristic polynomial of the transfer functions between the incoming disturbance input and the measured outputs available from PMUs. For example, consider a set of N PMUs. The following recurrence equation can be derived from the transfer functions:

$$\underbrace{\begin{bmatrix} y_i(n) \\ y_i(n+1) \\ \vdots \\ y_i(n+\ell) \end{bmatrix}}_{\mathbf{c}_i} = \underbrace{\begin{bmatrix} y_i(n-1) & \cdots & y_i(0) \\ y_i(n) & \cdots & y_i(1) \\ \vdots & & \vdots \\ y_i(n+\ell-1) & \cdots & y_i(\ell) \end{bmatrix}}_{H_i} \underbrace{\begin{bmatrix} -b_1 \\ -b_2 \\ \vdots \\ -b_n \end{bmatrix}}_{\mathbf{b}} \quad (1)$$

where $y_i(t)$ are the sample data available at time $t = 0, 1, \ldots, M$ for PMU i, $i = 0, 1, \ldots, N$. ℓ is an integer satisfying $n + \ell \leq M - 1$, where $M - 1$ is the time index of the most recent measurement. The problem is to compute the vector b, and thereafter solve the roots of the characteristic polynomial (whose coefficients are given by the entries of b) to obtain the desired oscillation modes.

We use linear regression to calculate the coefficients b, i.e., we calculate the b that results in the least sum of squares, as expressed in Equation 2:

$$\min_{\mathbf{b}} \frac{1}{2} ||H_i \mathbf{b} - \mathbf{c_i}||^{\mathbf{2}} \quad (2)$$

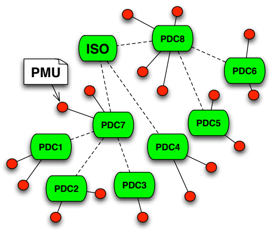

Fig. 4. PDC Tree Topology Example

Next, following [1], we enhance the centralized algorithm by replacing the centralized PDC with multiple distributed PDCs organized in tree topology. In the distributed algorithm, regional PMUs transmit a data stream to their local PDCs. PDCs at the same level in the tree topology estimate the oscillation modes using the data within their own domains. Thus, this problem becomes a *global consensus problem* over a network of N regional utility companies, which can be estimated via distributed protocols with one central independent system operator (ISO) performing a *supervisory* step to guarantee convergence. This ISO is the root PDC in the tree topology. Fig. 4 depicts the tree topology of distributed PDCs.

We use the ADMM algorithm [10] to solve this global consensus problem. To this end, PMUs and PDCs communicate in steps as follows:

1) Regional PMUs send raw data to local PDCs periodically. This transmission channel is marked as bold lines in Fig. 4.

2) Local PDCs update the local coefficients $\mathbf{b}_i$ via the calculations of Equations 1 and 2. PDCs send the estimation results to parent PDCs in the tree (e.g., $PDC1$, $PDC2$, and $PDC3$ send results to $PDC7$).

3) The central ISO-level PDC gathers the values $\mathbf{b}_i$ and broadcasts their mean $\bar{\mathbf{b}}$ back to regional PDCs. The mean coefficients are used in the next iteration of coefficient estimations at local PDCs.

ADMM guarantees that $\bar{\mathbf{b}}$ will eventually converge to the solution of the centralized problem. The desired eigenvalues can then be found by solving for the roots of the characteristic polynomial given by $\bar{\mathbf{b}}$.

B. Prony and ADMM using a Distributed Storage System

The Prony and ADMM algorithms described in Section III-A do not tolerate network failures. For example, estimation results in the region monitored by $PDC7$ in Fig. 4, which includes sub-regions monitored by $PDC1$, $PDC2$, and $PDC3$, could not be sent to the central PDC (C-PDC) at the ISO if the network channel between $PDC7$ and C-PDC fails. The coupling of PMUs and PDCs in a tree topology reduces its resilience. Thus, we enhance the algorithm by utilizing our distributed storage system as a middleware to transmit data. Fig. 3 illustrates the new architecture, where PDCs are used as storage nodes and organized as a Chord ring to construct a cloud storage service.

This distributed storage system decouples the strong dependency between PDCs in the tree topology. In the new architecture, each PDC maintains the IP addresses of a list of storage nodes. In case of a network channel failure, another storage node in the list can be picked as the gateway for requests to the cloud storage service. Since data in the storage system are distributed among storage nodes and redundant data are stored in different nodes, the storage system can increase the resilience of the Prony and ADMM algorithms.

In this scenario, the different steps of the ADMM algorithm are modified as follows: 1) PMUs send raw data to their local PDCs. 2) Local PDCs update the local coefficients $\mathbf{b}_i$ and send the estimation results to the storage system. 3) The C-PDC gathers the values $\mathbf{b}_i$ from the storage and sends their mean $\bar{\mathbf{b}}$ back to the storage. The C-PDC sets up a delay threshold (i.e., a deadline for data requests) when gathering the $\mathbf{b_i}$'s of all local PDCs. If the deadline expires before the C-PDC gets a $\mathbf{b_i}$ of a particular local PDC from the storage system, the previous $\mathbf{b_i}$ of that local PDC is used to calculate the mean $\bar{\mathbf{b}}$. (4) The mean coefficients $\bar{\mathbf{b}}$ are fetched by local PDCs to engage in the next iteration of coefficient estimations. We update the convergence condition in the centralized system to ignore the impact of using previous data, which always has a difference of zero. Equation 3 depicts the convergence condition. $\bar{\mathbf{b}'}$ is the average coefficient estimate of the previous iteration. Z is the number of new local PDC data that are used to calculate $\bar{\mathbf{b}}$. δ is the convergence threshold.

$$\frac{1}{Z}||\bar{\mathbf{b}} - \bar{\mathbf{b}'}||^{\mathbf{2}} \leq \delta \tag{3}$$

In addition, if the C-PDC fails, any one of the other PDCs can turn into a C-PDC to perform the supervisory step to guarantee convergence (as long as all PDCs agree on the same new C-PDC, which is known as the leader-election problem in Computer Science).

IV. Evaluation

In this section, we present experimental comparing the centralized measurement system with the resilient measurement system, where the latter uses the distributed storage. We inject network failures in our experiments to demonstrate the advantage of the distributed storage.

A. Experimental Setup

In our experiment, 4 nodes simulate local PDCs and 1 node simulates the central PDC. Each local PDC is associated with one PMU, which provides real-time data (sample rate 60Hz). This section provides the result of simulations running on Linux on bare metal machines.

In the direct channel mode, these five PDCs communicate directly via TCP (described in Section III-A). In contrast, in the distributed storage mode, our storage system utilizes the PDC nodes as the storage nodes to provide the cloud storage service (see Section III-B).

We inject two types of network failures. For the first type, we inject intermittent failures by probabilistically dropping the data on the TCP channel between the PDCs and the C-PDC for the direct mode and between the PDCs and the access points of the storage system for the distributed mode. For the second type, we inject long-term link failures by dropping the data on the link after certain iterations. In the direct link mode, the C-PDC has to use the historical data to calculate the new coefficients while in the storage mode, an alternative access point to the storage system provides the correct data. For distributed storage, data becomes unavailable only if all channels between a PDC and all access points of the storage system are broken.

We compare the convergence speed and, more importantly, the estimation accuracy of both modes when we inject probabilistic failures. The convergence speed is represented by the number of iterations that the algorithm performs to calculate the global minimum coefficients $\bar{\mathbf{b}}$. Fewer iterations indicate faster convergence. Then, we use the global minimum coefficients to calculate the eigenvalues of the state matrix with the Prony algorithm. The estimation accuracy is calculated as the relative error between the calculated eigenvalues of the state matrix and the actual eigenvalues (known as a priori in our evaluation). Since the injected network failures may cause the PDCs to use historical data instead of the recent data, the ADMM algorithm can converge at different minimum coefficients, which result in different estimation accuracies.

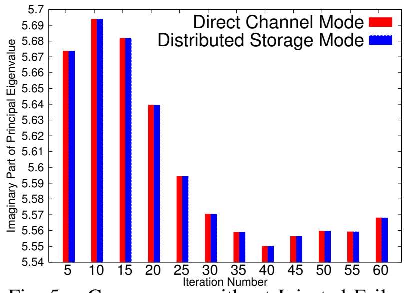

Fig. 5. Convergence without Injected Failures

Fig. 5 depicts the imaginary part of one principal eigenvalue in every 5 iterations when no failures were injected in the experiments. The red line (for direct channel mode) matches the blue line (for storage mode) and $\bar{b}$ converged in both modes after 63 iterations. When it converges, the imaginary part of the principal eigenvalue is 5.5684. The relative error is 0.072% (the actual value is 5.5644). In this experiment, we used a threshold value of 0.00012 to stop the iteration (see Eq. 3). When we injected failures (10% probability to break a channel in each communication), the convergence speed and accuracy in the storage mode, as shown in Fig. 6, stayed the same since the PDCs can utilize another entry points to the storage system to access data. By comparison, the direct channel mode converged earlier (after iteration 15) with the same stopping threshold as depicted by the black line. However, it resulted in an inaccurate estimation (with imaginary part of the principal eigenvalue 5.6974), which has a relative error of 2.390%. The accuracy of the direct channel mode decreases when failures where injected because of the data loss. When we decrease the stopping threshold for the direct channel mode, the number of iterations increases and the accuracy increases. However, as shown in the red line, it resulted in a less accurate estimation (relative error 3.127%) than the storage mode even when it required more iterations.

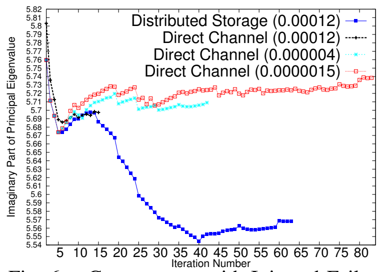

Fig. 6. Convergence with Injected Failures

Table. I presents the estimation accuracy when we injected long-term failures to different numbers of PDCs (stop threshold 0.00012). We calculate the relative error of the principal eigenvalue and the L_2 norm of all eigenvalues. The accuracy of the storage mode is not impacted while the direct channel mode has larger errors for long-term failures.

TABLE I
COMPARISON OF ACCURACY WITH LONG-TERM FAILURES

Mode	# Failures	Principal Eigenvalue Error	L_2-norm Error
Storage Mode	1	0.072%	0.105%
	2	0.072%	0.105%
	3	0.072%	0.105%
Direct Channel	1	2.237%	1.961%
	2	2.346%	2.071%
	3	2.448%	2.144%

V. CONCLUSIONS

In this paper, we present our resilient distributed software infrastructure for Wide-Area Measurement Systems. As the core component of our software infrastructure, our real-time distributed storage system can satisfy the network QoS requirements of WAMS with failure tolerance and self-recoverability. We integrated wide-area monitoring algorithms with our storage system to demonstrate the benefit that our storage system can provide to the oscillation modes estimation algorithm. As a trade-off, our storage system runs as a middleware between PDCs, which increases the time for a single communication. However, our experimental results show that even when network failures occur, the algorithms can use alternative access points of the storage system to store/obtain data. Without our storage system, the algorithms have to use historical data, which decreases the accuracy and speed of convergence.

REFERENCES

[1] S. Nabavi, J. Zhang, and A. Chakrabortty, "Distributed optimization algorithms for wide-area oscillation monitoring in power systems using interregional pmu-pdc architectures," *Smart Grid, IEEE Transactions on*, vol. 6, no. 5, pp. 2529–2538, 2015.

[2] Y. Xin, I. Baldine, J. Chase, T. Beyene, B. Parkhurst, and A. Chakrabortty, "Virtual smart grid architecture and control framework," in *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[3] T. Qian, A. Chakrabortty, F. Mueller, and Y. Xin, "A real-time distributed storage system for multi-resolution virtual synchrophasor," in *Power & Energy Society General Meeting*. IEEE, 2014.

[4] T. Qian, M. Frank, and Y. Xin, "A real-time distributed hash table," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*. IEEE, 2014, pp. 1–10.

[5] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM 2001*, San Diego, CA, September 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *in Proceedings of ACM SIGCOMM*, 2001, pp. 161–172.

[7] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, and J. Chase, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, 2012, pp. 97–113.

[8] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *ACM Symposium on Theory of Computing*, 1997, pp. 654–663.

[9] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2002, p. 213.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.