A Real-time Distributed Storage System for Multi-Resolution Virtual Synchrophasor

Tao Qian[1], Aranya Chakrabortty[2], Frank Mueller[1]
[1]Department of Computer Science, [2]Electrical Engineering
North Carolina State University
Raleigh, NC, USA
{tqian2,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*—With the continuing large-scale deployment of Phasor Measurement Units (PMU), the Wide-Area Measurement System (WAMS) technology is envisioned to evolve towards a distributed architecture where multiple sets of distributed Phasor Data Concentrators (PDCs) collectively process PMU data to achieve real-time distributed intelligence. Emerging applications developed under this vision will pose stringent but heterogeneous real-time requirements on throughput, delay, and reliability performance of the underlying communication and computing infrastructure. To address this problem, we present a novel virtual PMU (vPMU) architecture that decomposes phasor samples into multiple resolution layers. For a particular receiver with a certain resolution requirement, a complete set of PMU data can be composed by combining samples from the lower layers, without the need for samples from higher layers. We design and implement a real-time distributed storage system to support the virtual PMU data communication. We extend the Chord algorithm so that the response time of data communication can be bounded by our storage system. In addition, we use queuing theory to analyze the response time of requests with our stochastic model.**

I. Introduction

In recent years, PMUs have been successfully commercialized and deployed aggressively in many countries to support efficient wide-area measurement and control [1]. With the rapidly increasing number of PMUs and the resulting heterogeneous Quality of Service (QoS), security, and scalability requirement of emerging applications [2], the current state-of-art centralized data processing infrastructure of WAMS will no longer be tenable in a few years, and a decentralized architecture that supports distributed and autonomous intelligence will become imperative [3]. Accordingly, in our recent paper [4], we proposed a cloud computing based virtual smart grid (vSG) framework that allows dynamic creation of distributed applications in clouds to connect to the tailored set of PMUs in real-time. A critical concept in the vSG framework is the so-called *virtual PMU (vPMU)* that can simultaneously generate multi-resolution phasor measurement samples for WAMS applications that require different sample rates. In parallel, as the long as more PMUs are deployed and more WAMS applications require PMU data, reliable real-time data transmission from PMUs to applications becomes a challenge. In this paper, we aim to design a distributed data storage middleware that runs between PMUs and WAMS applications. Such a data storage system stores periodic PMU data and serves the data to applications in a real-time fashion. This distributed storage system could be a very attractive choice for PMU-PDC topology as it could use PDCs as storage nodes to store PMU samples, and WAMS applications could proactively obtain required data from PDCs directly instead of reactively waiting for the data from PMUs.

Our distributed data storage system, as a data service layer between PMUs and applications, provides two API services: *put(key, data)* stores the PMU *data* with *key* as its index; *get(key)* returns the PMU data associated with *key* in the system. Our idea is to utilize PDCs as storage nodes and build a distributed hash table (DHT). DHTs, e.g., Chord [5] and CAN [6] are well suited for this problem due to their superior scalability, reliability, and performance over centralized or even tree-based hierarchical approaches. However, there is a lack of research on real-time bounds for these DHT algorithms for end-to-end response times of requests. Timing analysis is the key to enable our storage system to provide real-time services.

In this paper, we extend the Chord DHT algorithm to adopt one cyclic executive as the scheduler on each storage node to execute real-time requests. In addition, by defining the periodic request patterns according to the realistic WAMS applications requirement, we develop a stochastic model to derive response time for our distributed storage system to serve requests. We evaluate our model by simulations, which demonstrate that our stochastic model can result in highly reliable response time bounds.

The remainder of the paper is organized as follows. Section II introduces the novel virtual PMU architecture. Section III presents our distributed storage system design and timing analysis model. Section IV presents our experimental evaluation. Section V concludes the paper with future work.

II. Virtual PMU and Multi-Resolution Synchrophasor Measurement

The proposed decentralized PMU-PDC system, as shown in Fig.1, consists of multiple dynamic communication groups, one per application. In this paper, we use the non-recursive phasor estimation problem as a simple example to illustrate the proposed vPMU implementation, indicating the typical data flow mechanisms between a vPMU and any other WAMS application. The phasor estimate is obtained by sampling the sinusoidal voltage $x(t)$ at a sampling frequency Nf_0, f_0 being the nominal frequency, where N is the sample size and $\theta = \frac{2\pi}{N}$ is the sampling angle in one cycle, and ε_n is a zero-mean noise process with a variance of σ. This can be represented in matrix notation as $\mathbf{x} = \mathbf{S} \times \mathbf{X} + \varepsilon$, and the weighted least-squares solution for phasor estimate is

$$\hat{\mathbf{X}} = [\mathbf{S^T W^{-1} S}]^{-1} \mathbf{S^T W^{-1} x} \tag{1}$$

where $\mathbf{W} = \sigma^{\mathbf{2}} \times \mathbf{I}$, and the standard deviation of the estimate error is $\frac{\sigma}{\sqrt{N}}$, which is inversely proportional to the sampling rate. We use the notations $\mathbf{x}(\mathbf{N})$ and $\mathbf{S}(\theta)$ to reflect different sampling resolutions within a sampling cycle (window).

Assuming that a PMU needs to send a set of phasor data with different sampling resolutions in a window, $\{N_1, N_2, \cdots, N_k\} \in \vec{N}$, to k applications, the PMU needs to be able to sample N_{max} per cycle, where N_{max} is the least common multiple of numbers in $\vec{N}$ and $\theta_{max} = 2\pi/N_{max}$. Without loss of generality, we assume $N_k = N_{max}$, $N_1 < N_2 < \cdots < N_{max}$ so that samples with resolution N_i (represented by the sample set $\vec{N_i}$) is a subset of samples with resolution N_{max} (represented by the sample set $\vec{N}_{max}$). $\vec{N_i}$ is formed by inserting an extra sample in the middle of every sampling interval of $\vec{N}_{i-1}$. Not counting the fixed first sample in $\vec{N_i}$, we get $N_i = 2 \times N_{i-1}$ and $\Delta N_i = N_{i-1}$. In Fig. 2, sampling with three different resolutions is depicted, where $N_{max} = N_3$.

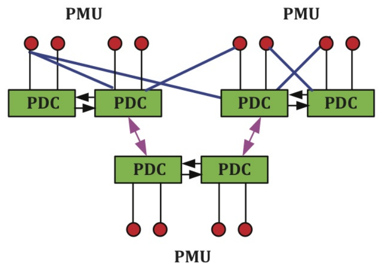

Fig. 1. Distributed PMU-PDC Communications

For the i^{th} application requiring resolution $N_i \in \vec{N}$, $\theta_i = \frac{N_{max}}{N_i} \times \theta_{max}$, the phasor estimate can be calculated by taking measurement $\mathbf{x}(\mathbf{N_i})$ and using the matrix $\mathbf{S}(\theta_\mathbf{i})$, generated by taking the rows out of $\mathbf{x}(\mathbf{N_{max}})$ and $\mathbf{S}(\theta_\mathbf{max})$ at a rate of $\frac{N_{max}}{N_i}$. Conceptually, each N_i represents a unique measurement resolution of a virtual PMU (vPMU).

From the networking perspective, each vPMU multicasts a phasor data train of window size N_i to the designated receivers in the i^{th} application. We observe that $\vec{N_i}$ can be constructed by combining $\vec{N}_{i-1}$ and $\Delta\vec{N_i} = \vec{N_i} - \vec{N}_{i-1}$. This naturally leads to a novel multi-layer phasor sampling scheme that decomposes the maximum (physical) sample train $\vec{N}_{max}$ into different layers starting from the minimum sampling rate N_1. For example, considering the maximum resolution sampling N_3 in Fig. 2, $\vec{N_1} = \{x_4, x_8\}$, $\Delta\vec{N_2} = \{x_2, x_6\}$, and $\Delta\vec{N_3} = \{x_1, x_3, x_5, x_7\}$. A higher layer can only be composed if all its lower layers are present, i.e., $N_i = N_1 + \sum_{j=2}^{i} \Delta N_i = 2^i \times N_1$, where $i = 2, \cdots, max$.

In addition to reducing PMU overhead and communications resources, this multi-resolution approach could tolerate PDC failures with the distributed storage system. In the distributed

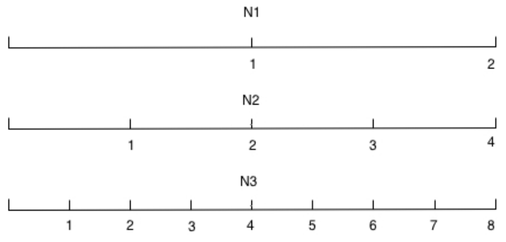

Fig. 2. Multi-Resolution Sampling

storage system, PMU samples are not only stored in the PDC that is connected with the PMU, but also duplicated at other PDCs in the cloud. As long as at least one PDC, which has the demanded data, is alive, the WAMS application could obtain the data via the storage service.

III. Distributed Storage System

A distributed storage abstraction provides support to store PMU data in a distributed manner, i.e., sets of PDCs and WAMS applications may obtain data from sets of PMUs through this layer. The layer builds on and complements the multicast multi-layer routing protocols in that it provides an additional protocol level that ensures predictable time bounds for reads/writes, resilience, and scalability. We utilize distributed hash tables (DHTs) as a means to realize this resilient and scalable storage abstraction to disseminate PMU data in a distributed manner required to state estimates. A network overlay will be given by a Chord-like ring with finger pointers, which is self balancing and self repairing [5]. Notice that this ring overlay may be mapped to multi-layer multicast routings as a means of optimizations or mapped to a cross-linked tree [7] as a means to implement it over a given physically or logical network topology. This ring structure is of higher cost to maintain than other overlay topologies but provides a natural way to orchestrate phasor estimates and, optionally, actions of disjoint PDCs based on key/value pairs. DHTs can store raw PMU data, memorize state estimates, multicast patterns and even actuation intentions.

In this section, we present the our extension to the Chord algorithm to provide real-time services and derive our timing analysis model.

A. Chord

The Chord algorithm manages storage nodes in a ring-like topology. Chord uses a consistent hashing algorithm [5], [8] to generate an identifier for each node by hashing the node's IP address, then orders the nodes by their identifiers. When a *put* or *get* request for a specific key is required, Chord uses the same hash function to encode the key. It then assigns the key to the first node on the ring whose identifier is equal to or follows the hash code of that key. This node is called the successor node of the key, or the target node of the key. Fig. 3 depicts an example of a Chord ring. It is sufficient to locate one's successor node by maintaining the nodes in a wrap-around circular list (ring) in which each node has a reference to its successor node. In the figure, *N10* finds the target node

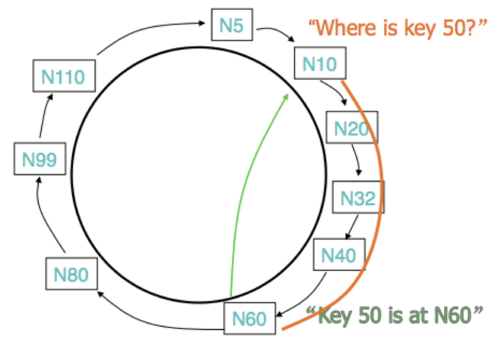

Fig. 3. Chord-ring example

N60 for *K50* by traversing a sequence of intermediate nodes *N20*, *N32*, *N40*.

This linear algorithm is not scalable with increasing numbers of PDCs. In order to reduce the number of intermediate nodes, Chord maintains the so-called finger table on each node, which acts as a routing table, to decide the next hop to forward lookup requests. For example, assuming the largest identifier for nodes on the ring in Fig. 3 is *128*, the finger table for *N10* is in Table I. Each entry in the table indicates one routing rule: the next hop for a given key is the successor node in the entry if its interval of that entry includes the key. For example, the next hop for *K50* is *N60* as *50* is in *[42, 74)*.

TABLE I
FINGER TABLE FOR *N10*

#	start	interval	successor
0	11	[11, 12)	N20
1	12	[12, 14)	N20
2	14	[14, 18)	N20
3	18	[18, 26)	N20
4	26	[26, 42)	N32
5	42	[42, 74)	N60
6	74	[74, 11)	N80

In general, a finger table has $log(N)$ entries, where N is the largest node identifier. For node k, the start of the i^{th} finger table entry is $k + 2^i$, the interval is $[k + 2^i, k + 2^{i+1})$, and the successor is the *successor* of key $k + 2^i$. Because of such way that finger tables are constructed, each step of a lookup request along finger pointers reduces the distance to the target node for a given key to at most half of the original distance relative to the ring structure [5]. As a result, the total number of visited nodes to serve a lookup request is at most $log(N)$, which is more scalable than the previous linear algorithm.

B. Real-time Chord

The Chord algorithm does not provide real-time bounds for lookup requests. In order to serve requests in a real-time fashion, we extend the Chord algorithm in two aspects: (1) We define the request pattern according to the realistic needs of WAMS applications; (2) We employ a cyclic executive [9] to schedule these requests on each storage node.

Request pattern. In WAMS applications, requests follow a periodic pattern. For example, PMUs periodically send real-time data to PDCs [10], so that PDCs issue *put* requests periodically. At the same time, PDCs or WAMS applications need to fetch data from other PDCs to monitor global power states and control the state of the entire power grid periodically, especially for wide-area control. As a result, PDC nodes execute periodic tasks to serve these requests. In addition, it requires a sequence of followup tasks to serve one request, as once the initial node issues a request, it requires a job on each intermediate node to forward the request according to its finger table until the request is eventually served. These followup-tasks are aperiodic, as we assume a non-constant network delay to forward the request on the Chord ring.

Cyclic executive. Our real-time Chord employs a cyclic executive to schedule these periodic and aperiodic tasks on each node. The cyclic executive divides the timeline into time frames. In each frame, it first executes periodic tasks that need to complete in that frame, and then executes aperiodic tasks. For example, assume a PDC with two PMUs (PMU-a and PMU-b) attached, each of which send PMU samples to the PDC at a 60Hz sample rate. Table II depicts the periodic tasks that the PDC needs to schedule to process these PMU samples and commit them to the storage system. In a hyperperiod of 50ms, three frames are included in this schedule table. After each hyperperiod, a cyclic executive wraps around to repeatedly schedule periodic tasks for the first frame of the next hyperperiod. Within a hyperperiod, the cyclic executive accepts timer interrupts at the beginning of frames to schedule corresponding periodic tasks according to its schedule table.

TABLE II
REQUEST PATTERN EXAMPLE

#	start time(ms)	end time(ms)	periodic tasks
0	0	16.7	receive, put(PMU-a), put(PMU-b)
1	16.7	33.4	receive, put(PMU-a), put(PMU-b)
2	33.4	50.0	receive, put(PMU-a), put(PMU-b)

In order to accept aperiodic jobs sent by other nodes over the network, each node schedules a periodic *receiving* task at the beginning of each frame, as in Table II. A receiving task accepts jobs that arrive in the previous frame and puts them into a FIFO queue. When the periodic jobs in one frame are finished, the cyclic executive utilizes the remaining time in that frame to execute aperiodic jobs obtained from the FIFO queue. A side effect of this receiving mechanism is that aperiodic jobs can only be scheduled in the next frame.

C. Timing analysis model

With the request patterns and the schedule mechanism, we use queuing theory to build a stochastic model that provides upper bounds for the response time of aperiodic jobs on one storage node (single-node response time), and the end-to-end response time of lookup requests, which is the aggregation of a sequence of single-node response times along the forwarding path. In our model, the response time consists of total execution time and wait time. We use a Poisson process to model arrivals of aperiodic tasks. Table III depicts the notation of our model. We also use the same notation without the subscript for the vector to denote all values. For example, U is $(U_0, U_1, \ldots, U_K)$, C is $(C_0, C_1, \ldots, C_M)$. In the table,

H, K, ν, U are known from the schedule table. M is defined in our DHT algorithm. C is obtained by measurement from the implementation. λ is calculated from request patterns.

TABLE III
NOTATION

Notation	**Meaning**
H	hyperperiod
K	number of receiving jobs in one hyperperiod
ν_i	time when the i^{th} receiving job is scheduled [1,2]
U_i	utilization of periodic jobs in time interval (ν_i, ν_{i+1})
M	number of different types of aperiodic jobs
λ_i	average arrival rate of the i^{th} type of aperiodic jobs
C_i	maximum execution time of the i^{th} type of aperiodic jobs
g_i	number of arrivals of the i^{th} aperiodic job
$E(x, g)$	execution time for aperiodic jobs that arrive in time interval of length x
$W(i, e)$	response time for aperiodic jobs, if they are executed after i^{th} receiving job
$P(n, \lambda, x)$	probability of n arrivals in time interval of length x, arrival is a Poisson process with arrival rate λ

[1] ν_i are relative to the beginning of the current hyperperiod.
[2] For convenience, ν_0 is defined as the beginning of the current hyperperiod, and ν_{K+1} is the beginning of the next hyperperiod.

Equation 2 formalizes the aperiodic jobs' execution time $E(x, g)$. Without loss of generality, we use notation E and $E(x)$ to represent $E(x, g)$.

$$E(x, g) = \sum_{i=1}^{M} C_i * g_i \; with \; probability \prod_{i=1}^{M} P(g_i, \lambda_i, x) \quad (2)$$

We further define A_p as the time available to execute aperiodic jobs in (ν_1, ν_{p+1}). Let $A_0 = 0$.

$$A_p = \sum_{i=1}^{p} (1 - U_i) * (\nu_{i+1} - \nu_i) \quad (3)$$

To model the response time $W(i, E)$, we find the index $p \in [i + 1, K]$, so that $E \in (A_{p-1} - A_{i-1}, A_p - A_{i-1}]$. Thus, the response time is the length between receiving jobs i and p plus the time after receiving job p to execute the remaining workload that is leftover after p. This remaining workload is finished in one frame and the response time can be calculated from the schedule table.

Given an aperiodic job J of execution time C_m arrives at time t relative to the beginning of the current hyperperiod, let $p+1$ be the index of the receiving job such that $t \in [\nu_p, \nu_{p+1})$. We derive the response time of this job in different cases.

(1) Periodic jobs that arrive before ν_p cannot be finished before ν_{p+1}, which means the $leftover\ workload\ LW(\nu_p) = E(\nu_p) - A_p > 0$. In this case, the response time of job J is the time to wait for the next receiving job at ν_{p+1} in addition to the time to execute the aperiodic job workload $LW(\nu_p) + E(t - \nu_p) + C_m$, which is $W(p + 1, E(t) - A_p + C_m)$, after the $(p + 1)^{th}$ receiving job as in Equation 4.

$$R(C_m, t) = (\nu_{p+1} - t) + W(p + 1, E(t) - A_p + C_m) \quad (4)$$

(2) Periodic jobs that arrive before ν_p can be finished before ν_{p+1}, $LW(\nu_p) \le 0$. The response time in this case is given by Equation 5.

$$R(C_m, t) = (\nu_{p+1} - t) + W(p + 1, E(t - \nu_p) + C_m) \quad (5)$$

This completes the stochastic model $R(C_m, t)$ for the single node response time of an aperiodic job of execution time C_m that arrives at t. The end-to-end response time of a request is the aggregation of three parts in our real-time Chord implementation: 1) the response time of the initial periodic job, which is known by the schedule table; 2) the total response time of subsequent aperiodic jobs for a lookup service on a maximum of $log(N)$ nodes [5], where N is the number of nodes in the Chord ring; and 3) the final aperiodic jobs to serve the communication between the initial node and the target node.

IV. Evaluation

In our evaluation, we simulate the request patterns according to the needs of a wide-area monitoring system. We use four nodes to simulate four PDCs in a local area within an interconnection area. The nodes are not synchronized to each other relative to their start of hyper-periods as such synchronization would be hard to maintain in a distributed system. In our simulation, each PDC has 10 PMUs attached. The data exporting rate of each PMU is taken as 60 Hz. Table IV depicts the periodic requests of three frames in one hyper-period. It is sufficient to evaluate our model by only executing *put* requests in the simulation since *get* requests are served with the same response time as *put* requests in our current real-time storage system implementation.

TABLE IV
REQUEST PATTERN IN THE SIMULATION

start time(ms)	**end time(ms)**	**periodic tasks**
0	16.7	receive, put(PMU-1), .., put(PMU-10)
16.7	33.4	receive, put(PMU-1), .., put(PMU-10)
33.4	50.0	receive, put(PMU-1), .., put(PMU-10)

Table V includes the parameter values for our timing analysis model in the simulation. The cyclic executive schedules a $sleep$ job once the periodic jobs in each frame have executed. This $sleep$ job is to simulate the periodic computation on PDCs for power state estimation. As a result, the utilizations of periodic jobs in the three frames are $40\%, 40\%, 40\%$, respectively. Any put/get requests forwarded to other nodes in the DHT result in aperiodic (remote) jobs. We use $sleep$ again to simulate an execution time of $0.2ms$ for aperiodic jobs, which includes the message transmit time between the simulated nodes. In this experiment, the utilizations of periodic jobs and aperiodic jobs are 40% and 48%, respectively. The system is stable as the total utilization is less than 100%.

TABLE V
MODEL PARAMETER VALUES

Name	**Value**
H	50*ms*
K	3
ν	(0*ms*, 16.7*ms*, 33.4*ms*)
U	(40%, 40%, 40%)
M	1
λ	2.4/*ms*
C	0.2*ms*

Fig. 4 depicts the cumulative distributions of single-node response times for aperiodic jobs. The figure shows that 99.4%

of the aperiodic jobs finish within the next frame after they are issued under this workload, i.e., their response times are bounded by $33.4ms$. Our model predicts that 99.5% of aperiodic jobs complete within the $16.6ms$ deadline, which is a good match. In addition, for most of the response times in the figure, our model predicts that a larger fraction of aperiodic jobs are finished within the response times than the fraction in the experimental data, as the red curve for modeled response times is above the blue curve. This indicates that our model is conservative for this workload.

Our model underestimates response times of a part of aperiodic jobs as shown in the figure between $12ms$ and $18ms$. This is due to the fact that aperiodic jobs are released more frequently at the beginning of periodic job execution phase, e.g., $(0ms, 4ms)$, and aperiodic job execution phase, e.g., $(6.7ms, 12.0ms)$, in each frame. As a result, the arrivals of aperiodic jobs are clustered at the beginning of the next frame instead of following a Poisson distribution over the whole frame. However, considering the non-constant network delay in real WAMS communication environments, it is more suitable to model the arrivals as a Poisson process. In our future work, we will conduct experiments with real PMU data to confirm this.

Fig. 5 depicts the cumulative distributions of end-to-end response times for requests, i.e., the time between a put/get request and its final response after propagating over multiple hops (nodes) within the DHT. 97.3% of the requests finish within the $(75ms, 80ms)$ interval after their arrival. These requests require four aperiodic jobs on average and most of the aperiodic jobs (88.9%) are finished within an $(18ms, 22ms)$ interval after being issued. All requests finished within $90ms$ in the experiment, which was predicted by our model with a probability of 98.9%.

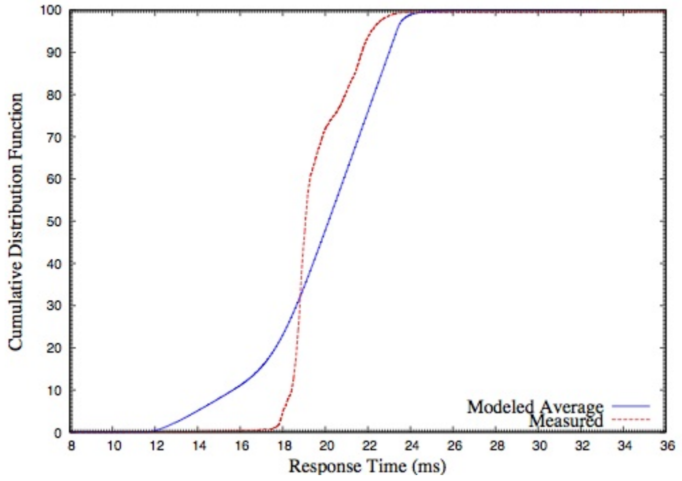

Fig. 4. Single-node response times distribution

V. Conclusions and Future Work

In this paper, we present a novel virtual-PMU architecture that can simultaneously generate multi-resolution phasor measurements from one physical PMU. To further enhance the performance and efficiency of disseminating Synchrophasor data, a distributed storage system is designed and implemented. It supports heterogeneous wide-area monitoring and control applications using a distributed PMU-PDC architecture. Our distributed storage system on the basis of Chord protocol provides real-time data communication between PMUs, PDCs and WAMS applications such as oscillation monitoring, special protection schemes, state estimation, and damping control. We also develop a stochastic model to analyze the response time of requests served by our distributed storage system. In the future, extensive performance studies will be conducted to quantify the performance gains over more traditional approaches. This includes studying the tradeoffs under different scheduling schemes for the abovementioned WAMS applications.

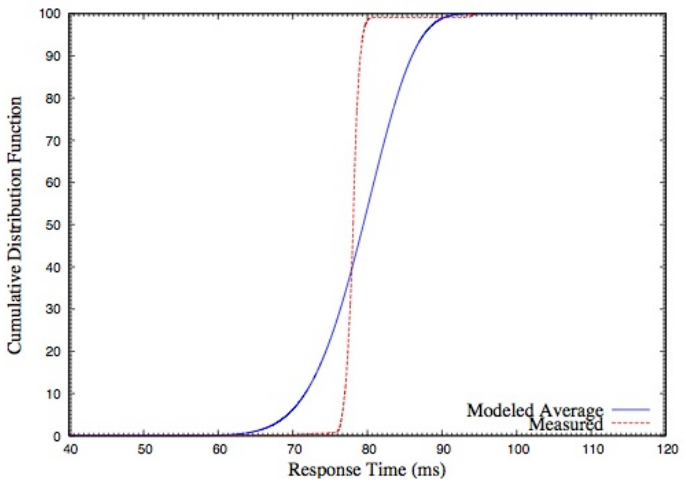

Fig. 5. End-to-end response times distribution

References

[1] J. D. L. Ree, V. Centeno, J. S. Thorp, and A. G. Phadke, "Synchronized phasor measurement applications in power systems," *IEEE Transactions on Smart Grid*, vol. 1, no. 1, June 2010.

[2] D. E. Bakken, A. Bose, C. H. Hauser, D. E. Whitehead, and G. C. Zweigle, "Smart generation and transmission with coherent, real-time data," *Proceedings of the IEEE*, vol. 6, no. 99, June 2011.

[3] A. Chakrabortty, "Handling the data explosion in tomorrow's power systems," *IEEE Smart Grid Newsletter*, Sep. 2011.

[4] Y. Xin, I. Baldine, J. Chase, T. Beyene, B. Parkhurst, and A. Chakrabortty, "Virtual smart grid architecture and control framework," in *2nd IEEE International Conference on Smart Grid Communications*, Oct. 2011.

[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] C. Zimmer and F. Mueller, "Fault tolerant network routing through software overlays for intelligent power grids," in *International Conference on Parallel and Distributed Systems*, Dec. 2011, pp. 542–549.

[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] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[10] W. A. Mittelstadt, P. E. Krause, P. N. Overholt, D. J. Sobajic, J. F. Hauer, R. E. Wilson, and D. T. Rizy, "The doe wide area measurement system (wams) project demonstration of dynamic information technology for the future power system," in *EPRI Conference on the Future of Power Delivery*, 1996.