Combining Tensor Decompositions and Graph Analytics to Provide Cyber Situational Awareness at HPC Scale

James Ezick,
Tom Henretty,
Muthu Baskaran,
Richard Lethin
Reservoir Labs, Inc.
New York, NY 10012
Email: {ezick, henretty, baskaran, lethin}
@reservoir.com

John Feo
Pacific Northwest National Laboratory
Richland, WA 99354
Email: john.feo@pnnl.gov

Tai-Ching Tuan,
Christopher Coley
Laboratory for Physical Sciences
College Park, MD 20740
Email: {tctuan, coleyc}
@lps.umd.edu

Leslie Leonard,
Rajeev Agrawal,
Ben Parsons
Information Technology Laboratory
U.S. Army Engineer Research and Development Center
Vicksburg, MS 93180
Email: {leslie.c.leonard, rajeev.k.agrawal, ben.s.parsons}
@erdc.dren.mil

William Glodek
BreakPoint Labs
Falls Church, VA 22042
Email: wglodek
@breakpoint-labs.com

***Abstract*—This paper describes MADHAT (Multidimensional Anomaly Detection fusing HPC, Analytics, and Tensors), an integrated workflow that demonstrates the applicability of HPC resources to the problem of maintaining cyber situational awareness. MADHAT combines two high-performance packages: ENSIGN for large-scale sparse tensor decompositions and HAGGLE for graph analytics. Tensor decompositions isolate coherent patterns of network behavior in ways that common clustering methods based on distance metrics cannot. Parallelized graph analysis then uses directed queries on a representation that combines the elements of identified patterns with other available information (such as additional log fields, domain knowledge, network topology, whitelists and blacklists, prior feedback, and published alerts) to confirm or reject a threat hypothesis, collect context, and raise alerts. MADHAT was developed using the collaborative HPC Architecture for Cyber Situational Awareness (HACSAW) research environment and evaluated on structured network sensor logs collected from Defense Research and Engineering Network (DREN) sites using HPC resources at the U.S. Army Engineer Research and Development Center DoD Supercomputing Resource Center (ERDC DSRC). To date, MADHAT has analyzed logs with over 650 million entries.**

I. Introduction

Tensor decompositions have been shown to isolate coherent patterns of potentially hostile behavior from within complex network traffic logs. This represents a new paradigm of network threat identification [1]. This pattern-based approach provides an advantage over classical signature-based threat identifications in that patterns can immediately link together multiple discrete activities separated by time, entity, or location in multidimensional data and can embody interactions that cannot be expressed (or often even anticipated) by rule signatures. As an unsupervised approach, tensor decomposition methods do not require training and have the potential to identify emergent behavior for which signatures have not yet been developed. Tensor methods provide a path to the discovery of deeper, higher-dimensional patterns at longer timescales – an enabler for detection of Advanced Persistent Threats (APTs) – a holy-grail-level problem in network cyber situational awareness.

Tensor decompositions are based on matrix operations extended to higher dimensions and have been greatly accelerated for both large shared- and distributed-memory architectures through algorithmic advances [2, 3] and the application of specialized sparse data structures [4, 5]. This emphasis on accelerating tensor decompositions for sparse data has made the technique practical for real datasets extending now into the billions of nonzero entries [6, 7]. These algorithms and data structures provide a foundation to apply tensor decomposition methods to cyber data at massive scale. This expansion of capability creates the potential to detect more malicious behavior across a wider variety of data sources and a larger range of time scales than any prior analysis.

Sponsored by the United States Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) as part of the Applicability of HPC to Cyber Situational Awareness (SA) initiative under contract W81EWF80232712.

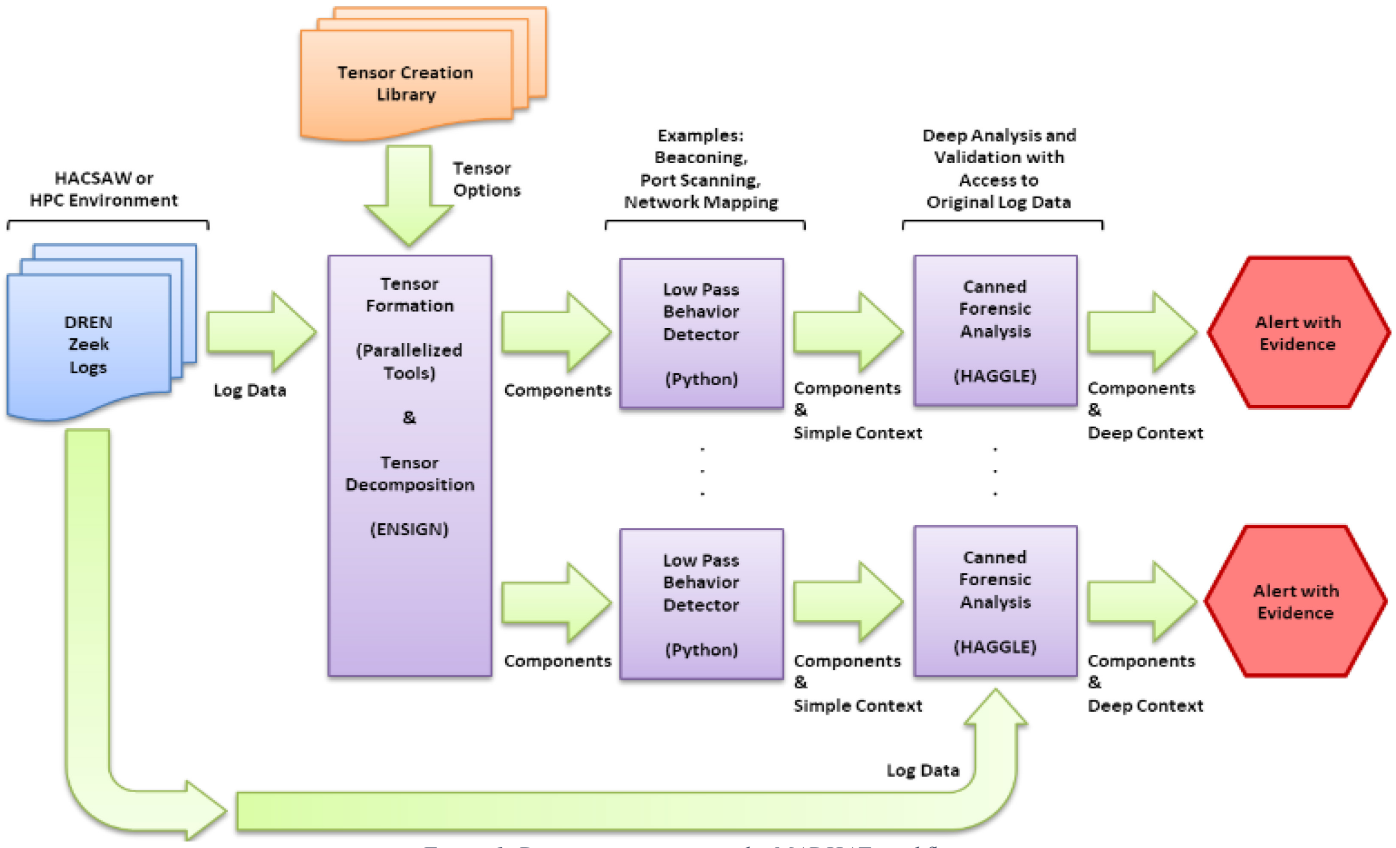

Tensor Creation Library
Tensor Options
HACSAW or HPC Environment
DREN Zeek Logs
Log Data
Tensor Formation (Parallelized Tools) & Tensor Decomposition (ENSIGN)
Components
Examples: Beaconing, Port Scanning, Network Mapping
Low Pass Behavior Detector (Python)
Components & Simple Context
Deep Analysis and Validation with Access to Original Log Data
Canned Forensic Analysis (HAGGLE)
Components & Deep Context
Alert with Evidence
Log Data

Figure 1. Diagram representing the MADHAT workflow.

To truly realize this vision, it is also necessary to automate the analysis of the patterns isolated by tensor decompositions. At hundred-million scale and beyond, it makes sense to perform decompositions that result in one hundred or more component patterns for a single tensor. Furthermore, there is significant value gained by decomposing multiple tensors formed from different aspects of the data, revealing different behaviors. As a result, the number of component patterns that need to be analyzed explodes, and the workload of individually examining each resulting pattern on a daily basis quickly becomes unrealistic. Large graph data structures are a natural choice for representing linked metadata at scale and offer rich query capabilities that have also been optimized for high-performance computing environments [8]. As a first-line analytics tool, however, graph-based approaches can suffer from the "boil-the-ocean" problem of having to search the totality of massive graphs to find instances of specific, sometimes complex subgraphs, among potentially billions of interconnected records.

MADHAT (Multidimensional Anomaly Detection fusing HPC, Analytics, and Tensors) [9] is based on the insight that large-scale tensor decompositions can be used to create an effective roadmap for targeted graph queries that confirm or reject behavior hypotheses. MADHAT combines ENSIGN [10] high-performance tensor decompositions with HAGGLE [11] parallelized graph analytics into a unified data science workflow supporting data preparation, analysis, reflection, and dissemination with both scriptable and visual components. Data sources are ingested into ENSIGN and tensor decompositions are performed. A coarse-grained triage of the resulting patterns identifies activities requiring deeper, targeted inspection. These patterns are passed to automated forensic analyses driven by HAGGLE to validate the hypothesis that the identified activities are of concern. These analyses are capable of combining elements of the original network log data with contextual information including network topology, preferences and priorities, whitelist and blacklist information, and external information such as published alerts. This contextual information allows adaptation and customization to specific environments that have the potential to improve the quality of resulting alerts while also reducing alert clutter.

MADHAT is being developed using resources made available by the HPCMP (High Performance Computing Modernization Program). This includes development support in their HACSAW (HPC Architecture for Cyber Situational Awareness) environment. This environment provides access to a variety of modern data science tools inside a Python-based Jupyter environment layered on top of access to months of data collected from the DREN (Defense Research and Engineering Network). In particular, this effort leverages Zeek (formally Bro) Intrusion Detection System (IDS) [12] network connection logs collected from various US DoD sites. Zeek IDS connection logs capture both inbound and outbound communications to DREN sites and individual records include more than twenty (20) fields with valuable metadata. The DREN itself encompasses more than 100 individual sites.

This paper describes an on-going effort to demonstrate the value of leveraging HPC to address several critical needs in cyber situational awareness:

- **Need for non-signature based detection**

  Signature based techniques are not adequate in operational cyber environments because it is not always feasible to describe normal and abnormal network behavior up front.

- **Need for multidimensional analysis of cyber data**

  Cyber data is always associated with multiple attributes (for example, a network connection log has metadata attributes, such as timestamp, sender IP, receiver IP, receiver port, number of response bytes, etc.).

- **Need for an approach that scales to big data**

  The potential of identifying threats, attacks, and other anomalous behaviors in networks increases when the analysis is done over huge volumes of data collected from multiple sources and over a long period of time.

- **Need to reduce analyst cognitive load**

  False-positive clutter is a limiting factor in the adoption of new approaches; it is infeasible and unreliable to rely on human interpretation of a large volume of complex, multidimensional patterns.

The remainder of this paper is organized as follows. Section II outlines the MADHAT architecture and workflow. Experimental results to date are captured in Section III. Section IV provides conclusions, mentions related work, and describes directions for future work.

II. Architecture and Workflow

This section describes the MADHAT workflow illustrated in Figure 1. The workflow consists of four stages:

1. Tensor Construction
2. Tensor Decomposition
3. Low Pass Behavior Detection
4. Canned Forensic Analysis

A. Tensor Construction

Tensor construction is the process of creating a tensor suitable for decomposition from a data source. At this stage of development, the experimental data source has been Zeek connection log (*conn.log*) files collected from DREN sites. Connection logs provide a record of each network connection and include approximately twenty fields organized in a row-column format similar to a CSV file, with one record per line. In the general MADHAT workflow, different tensors can be constructed from different combinations of logs and record fields stored in a Tensor Creation Library. For the examples used in this paper, a single tensor type is used that includes five fields from the connection log as dimensions of the tensor:

1. Timestamp (*ts*)
2. Originating IP address (*id_orig_h*)
3. Destination IP address (*id_resp_h*)
4. Responding port (*id_resp_p*)
5. Site

These fields were selected to allow for the discovery of broad patterns of point-to-point activity over time and across sites in message traffic.

Tensor construction requires that each dimension be independently binned and indexed, that duplicate rows (if any) be consolidated, and that the tensor be output in a common sparse tensor format along with associated maps translating numerical indices back to data labels. For the examples in this paper, time is binned by minute and the other dimensions are binned by their natural discrete values (that is, one index for each IP address, port, and site). To support tensor construction, the

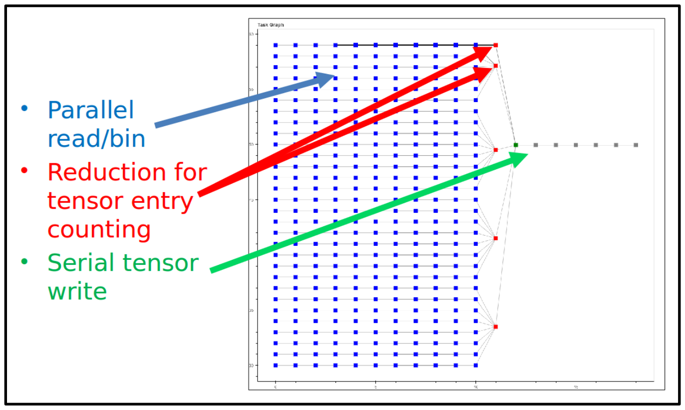


Figure 2. Diagram illustrating task parallelism in the tensor creation tool.

MADHAT workflow uses a custom tensor construction utility that leverages the Python Dask library to parallelize the reading, binning, and, indexing steps. The utility was evaluated on various small datasets and shown to produce speedups of 5.5 to 8.5x versus single-threaded performance on a 12 CPU machine (in real terms, from ~3 minutes to ~30 seconds on a dataset with 2 million entries). This speedup is significant because one day of DREN data can consist of over seven billion log entries that must eventually be converted to a tensor in an HPC environment. The tool was able to saturate all available CPU resources for a large proportion of its execution time due to embarrassingly parallel sections of data input and binning. This is shown in Figure 2.

B. Tensor Decomposition

MADHAT uses ENSIGN to perform tensor decompositions using the CP-APR [13] non-negative Poisson-based decomposition method. This method has been optimized within ENSIGN for HPC environments using a hybrid MPI+OpenMP parallel implementation, with MPI used between nodes and OpenMP used within each multicore node. The implementation addresses a number of critical challenges introduced from the irregularity of the structure of sparse tensors [14] (Table 1).

The output of the tensor decomposition method is a pre-selected number of components that, in sum, approximate the original tensor. An annotated example component is shown in Figure 3. A component includes a score between 0 and 1 (y-axis) for each index (x-axis) in each dimension of the tensor and a weight that captures the prevalence of the represented pattern in the tensor. As an approximation, each combination of spikes (one per dimension) is indicative of a nonzero value in the original tensor (indicating a record in the original log data). The component in Figure 3 captures what appears to be a scan of a range of ports targeted at a machine at a US service academy.

Table 1. Critical challenges in decomposing sparse tensors.

Challenge	Approach
Load-balanced parallel execution	Light-weight load distribution (at the beginning of the decomposition)
Communication minimization	Selective tensor partition (distribution) to minimize communication volume and frequency Factor matrix (aka decomposition output) replication on selective modes to reduce communication frequency
Reduced memory footprint	Selective re-computation of intermediate data (vs. storing large footprint intermediate data)
Minimal computations	Efficient sparse tensor data structures to facilitate memory- and operation-efficient computations
Data locality	Fusion of computations to increase thread-local operations with improved locality

C. *Low-Pass Behavior Detection*

At scale, it is not practical to individually annotate each decomposition component. Detectors are a mechanism for quickly identifying components that require deeper analysis. In the MADHAT workflow, each detector is a Python routine that accepts a single component and determines whether a specific characteristic behavior is present in that component. The detectors are based on heuristics and operate exclusively on the scores contained within the component without any additional context. In this way, the cross-product operation of applying all detectors to all components is embarrassingly parallel. As examples, we have experimented with three basic detectors in the MADHAT workflow (Table 2).

Table 2. Basic detectors in the MADHAT workflow.

Detector	Purpose	Impact
Port Scan	Identify contact with a range of ports on a single target machine	Port scanning behavior found in nearly all DREN decompositions
Network Mapping	Identify contact with a range of destination IP addresses	Small number of machines repeatedly detected performing lateral scanning
Beaconing	Identify instances of periodic communication	Low detection rate within DREN decompositions

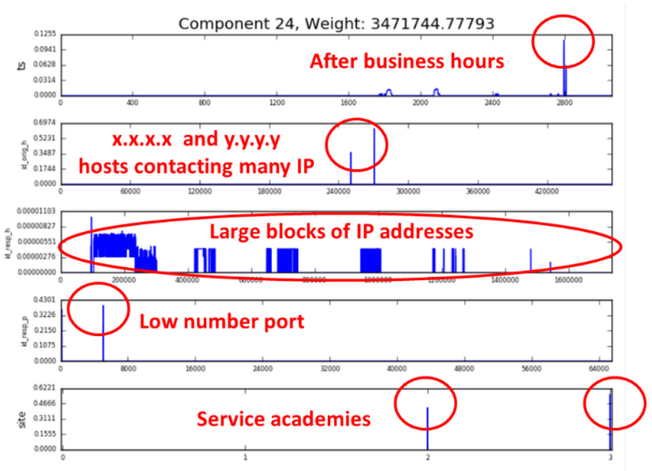


Figure 3. A single tensor decomposition component capturing a network mapping activity at two US service academies.

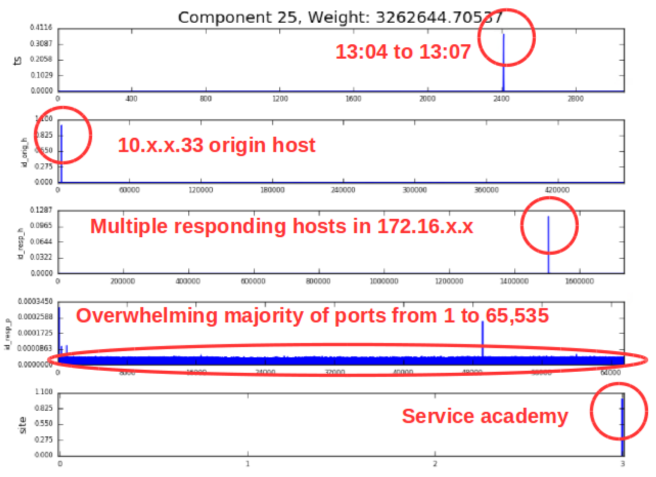


Figure 4. A single tensor decomposition component capturing a port scan activity at a US service academy.

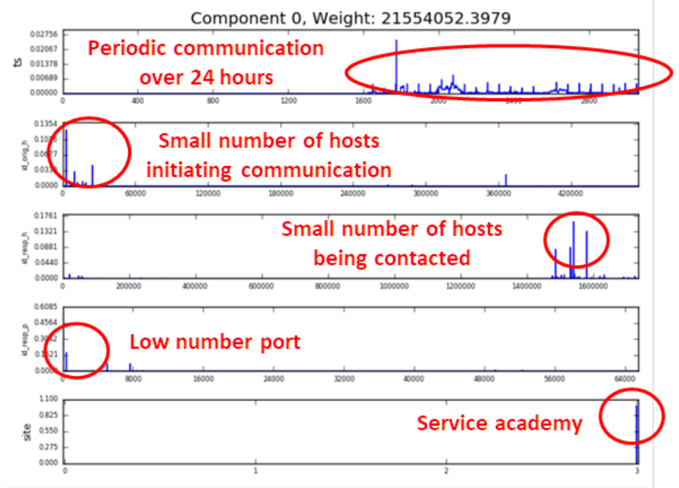
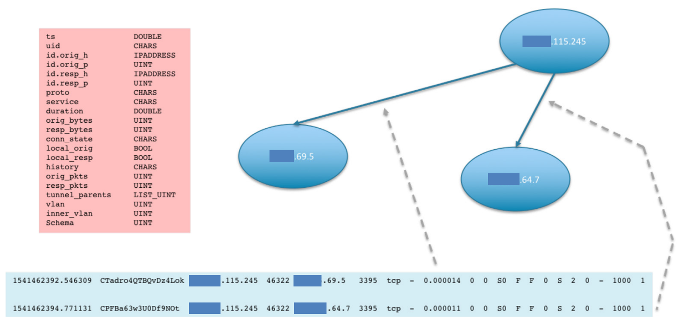


Figure 5. A single tensor decomposition component capturing a beaconing activity at a US service academy.

Port scanning and network mapping can be indicative of reconnaissance on a network. Reconnaissance activities attempt to identify specific nodes that can be attacked directly, through lateral movement, or possibly through exploitation of application-specific vulnerabilities. Beaconing can be indicative of surreptitious periodic behavior such as malware contacting a host. All three of these activities are characterized by patterns of messages that, in isolation, would likely be considered benign. Tensor decompositions tend to capture these behaviors as discrete components. Figure 3 illustrates an annotated component exhibiting port scanning behavior. Figure 4 illustrates an annotated component exhibiting network mapping. Figure 5 illustrates an annotated component exhibiting beaconing.

D. Canned Forensic Analysis



```
ts              DOUBLE
uid             CHARS
id.orig_h       IPADDRESS
id.orig_p       UINT
id.resp_h       IPADDRESS
id.resp_p       UINT
proto           CHARS
service         CHARS
duration        DOUBLE
orig_bytes      UINT
resp_bytes      UINT
conn_state      CHARS
local_orig      BOOL
local_resp      BOOL
history         CHARS
orig_pkts       UINT
resp_pkts       UINT
tunnel_parents  LIST_UINT
vlan            UINT
inner_vlan      UINT
Schema          UINT
```

```
1541462392.546309  CTadro4QTBQvDz4Lok  [redacted].115.245  46322  [redacted].69.5  3395  tcp  -  0.000014  0  0  S0  F  F  0  S  2  0  -  1000  1
1541462394.771131  CPFBa63w3U0Df9NOt   [redacted].115.245  46322  [redacted].64.7  3395  tcp  -  0.000011  0  0  S0  F  F  0  S  2  0  -  1000  1
```

Figure 6. An example of the HAGGLE hybrid data view.

For each low-pass behavior detector, there is an associated Canned Forensic Analysis (CFA) that processes components, one at a time, identified as exhibiting the behavior of interest specific to that detector. The goal of the CFA is to perform a deeper analysis that uses the identified component as a map to make targeted queries into the original data and relevant associated metadata. Associated metadata can include domain knowledge, network topology information, whitelist and blacklist information, and user preferences and priorities. While CFAs can vary in complexity, based on the sophistication of the analysis being performed, the overall structure of the CFA is that of a rule-based decision process – each query leads to zero or more follow-up queries until eventually a terminal state is reached that either does or does not raise an alert.

MADHAT uses HAGGLE (Hybrid Attributed Generic Graph Library Environment) to represent Zeek connection log data and associated metadata as a single coherent data structure. HAGGLE is a scalable platform that supports a variety of mixed analytic workloads, distributed, thread-safe data structures, and an abstract runtime layer that allows rapid porting of the software stack to modern conventional, distributed high-performance computing systems with accelerators as well as novel, purposely designed hardware.

The HAGGLE APIs were used to implement a hybrid data view that stores raw Zeek log data as a relational table and then builds a graph of servers (vertices) and connections (edges). An example is illustrated in Figure 6.

For each of the Port Scan, Network Mapping, and Beaconing detectors, the ENSIGN/detector front-end passes the times and entities of interest as parameters to the CFA. Each CFA then follows a similar pattern. Understanding that a tensor decomposition is an approximation of the original data, the CFA first validates the existence of the identified pattern in the original data. These tests typically involve nested for loops over entities, and edges from those entities, in the HAGGLE representation. Beyond existence checks, additional counting is performed and comparisons are made against connection whitelists or other thresholds such as message size or duration that might exclude the behavior from concern. Once the pattern is validated and a determination is made that it should not be excluded as a false-positive, an alert is generated with supporting context.

Expressing the forensics as nested parallel loops maximizes concurrency resulting in an oversubscription of threads to cores. The runtime system then exploits the oversubscription to hide long latency operations by multithreading.

III. Experimental Results

This section focuses on the performance of ENSIGN and HAGGLE in the context of the MADHAT workflow. Experiments with elements of the MADHAT workflow were performed using the Topaz cluster located at the ERDC DSRC. Topaz is an SGI ICE X System comprising 3,456 standard compute nodes. Each standard compute node has two 2.3-GHz Intel Xeon Haswell 18-core processors (36 cores) and 128 GBytes of DDR4 memory. Compute nodes are interconnected by a 4x FDR InfiniBand Hypercube network, and have Intel's Turbo Boost and Hyper-Threading Technology enabled. Memory is shared by cores on each node, but not between nodes. The experiments did not use the large compute node or GPU accelerated capabilities available on Topaz.

A. ENSIGN

The tensor decomposition method used in MADHAT is an iterative convergence algorithm. An immediate benefit of running in the HPC environment was the ability to explore the tradeoff between iteration count and final fit (quality of approximation) for a selection of large cyber data tensors

Table 3. Tensors formed from DREN Zeek conn.log data.

Tensor	**Number of Nonzeros**	**Dimension Sizes**	**Dates**

Daily - 4 DREN sites, 4 sites per tensor, one week of data (7 tensors)	~70M to ~258M	ts=1,531 to 3,603 (minutes)
		id.orig_h=~322K to ~595K
		id.resp_h=~399K to ~679K
		id.resp_p=~64K to ~65K
			11/5/2018 to 11/11/2018

Weekly - 4 DREN sites, one site per tensor, one week of data (4 tensors)	~673K to ~456M	ts=160 to 265 (hours)
		id.orig_h=~11K to ~1.2M
		id.resp_h=~27K to ~922K
		id.resp_p=~64K to ~65K
			11/5/2018 to 11/11/2018

Table 5. Example decompositions showing how reducing iteration count improves performance with minimal loss of precision.

Tensor	NNZ	Rank	Iters	Time	Final Fit	Delta Fit	Speedup	Cores
4 sites (1 day)	227.9M	100	100	5,867s (~1h38m)	0.375	0	1x	1800
			10	604s (10m4s)	0.352	-5.99%	9.73x	
1 site (1 week)	465.9M	100	100	29,996s (~8h20m)	0.786	0	1x	
			10	2,957s (49m17s)	0.783	-0.40%	10.14x	
			5	1,473s (24m33s)	0.776	-1.27%	20.36x	

ranging from ~70M to ~456M nonzeros (Table 3). The largest tensor was formed from more than 650M log entries.

From these tensors, over seventy (70) decompositions were performed on Topaz for performance tuning, cyber analysis, and cyber security experiments. The experiment consistently demonstrated that reducing the iteration count by 10-20x (a close approximation for runtime reduction, modulo I/O and initialization) resulted in only a minor reduction in final fit (Table 4).

A single tensor was used to demonstrate speedup on Topaz vs. a 20-core shared-memory segment of the HACSAW environment. The result was an end-to-end speedup of ~7.5x (Table 5). This included a speedup of ~20x of the core Matrix-Times-Tensor Khatri Rao Product (MTTKRP) computation that dominates large tensor decompositions (other factors affecting runtime include I/O operations and per-iteration convergence tests). The MTTKRP speedup is in line with expectations for a computation over a sparse tensor data structure that does not afford the same data locality as its dense analog. In total, this result demonstrates how HPC resources can reduce a daily decomposition to an execution time that is tractable for a nightly run.

Table 6. Using HPC resources makes large decompositions tractable for nightly runs.

Tensor	**Number of Nonzeros**	**Dimensions Sizes**	**Date**

Daily - 4 DREN sites, 4 sites per tensor, one week of data (1 tensor)	159.7M	ts=3,065
		id.orig_h=476K
		id.resp_h=1.73M
		id.resp_p=65K
		site=4
			5/1/2018

Platform	**Rank**	**Time**	**CPUs**
HACSAW	100	59,656 seconds (~16.5 hours)	20
HPC (Topaz)	100	7,965 seconds (~2.2 hours)	900

B. HAGGLE

The Port Scan and Network Mapping forensics were tested on a Zeek *conn.log* data file of 306M records. Data ingestion was parallelized by dividing file reads among nodes, having each node read and process its section, update a shared index pointer, and then move the processed data into a preallocated table. A property graph of servers (vertices) and Zeek records (edges) was created from the Zeek table data. The record's *id.orig_h* and *id.resp_h* attributes served as the source and destination vertices of an edge. The constructed graph had 703K vertices and 306M edges. Read and construction execution time and memory footprint on 8, 12, 16, and 24 nodes was measured. HAGGLE stores all data in memory to avoid slow disk accesses, requiring us to add additional compute nodes as data size grows. Since the constructed graph did not fit comfortably on 4 nodes, we ran on configurations of 8 nodes or more. The almost linear speedup shows that HAGGLE performance scales as data sizes grow (Table 6).

Table 4. Performance of HAGGLE forensics.

Node	**8**	**12**	**16**	**24**
Read	191.72s	130.80s	100.61s	71.15s
Construction	160.91s	106.17s	88.87s	64.05s
Footprint	40%	30%	21%	16%

Log files from one day of activity at a DREN site (over 150K network connections) were converted to a tensor (approximately 100K nonzero entries) and decomposed into 100 components. A run of the Network Mapping and Ports Scan low pass filters detected three (3) potential network mapping attempts and five (5) potential port scans. Unique destination IP addresses and destination ports were counted per component, where any IP address or port with a score over 0.00001 was added to the respective count.

Counts that exceeded thresholds of 100 unique destination IP addresses or 10 unique destination ports on a single IP address were flagged as a network map or port scan detection, respectively. Associated IP addresses, ports, and timestamps were handed off to HAGGLE for further analysis.

HAGGLE analysis of the network map detections confirmed the presence of two out of three detections in the original Zeek logs, while analysis of the port scan detections confirmed the presence of four out of five detections in the original Zeek logs. Further HAGGLE analysis determined the port scan detections to be TCP SYN scans, where a TCP connection was initiated by the scanning host but the TCP handshake is left incomplete.

Both the Port Scan and Network Mapping forensics ran in under a hundredth of second, so they had too little work to exhibit speedup as we increased the number of processors. The approach is clearly scalable to much larger datasets.

IV. Conclusion

MADHAT explores the idea that tensor decompositions can be used to provide a roadmap that allows the generation of focused, well-informed queries on large data. This approach combines the unique strengths of tensor methods as an unsupervised pattern-rather-than-signature-focused analytic with recent advances in representing and navigating large-scale linked metadata in graph form. The resulting workflow permits detection of unique patterns followed by fast forensic investigation that both validates an initial hypothesis and collects supporting context. MADHAT provides a path toward an operational workflow that shows promise in reducing both clutter and overall cognitive load on an analyst.

Some existing work in the literature (for example, MultiAspectForensics [15] and MalSpot [16]) has applied tensor decompositions to network traffic data in order to extract anomalies and malicious patterns. Unlike this other work, MADHAT extends beyond theoretical research and toolbox development to demonstrate a practical, scalable workflow. Prior work has investigated the additional step of automating alert analysis on DREN data using artificial neural networks [17]. Several commercial approaches are exploring the possibilities for enhanced cyber situational awareness created from the convergence of large data stores (for example, Chronicle Backstory [18]). The uniqueness of MADHAT lies in the application of tensor methods to navigate and extract value from large data stores.

The MADHAT workflow is extensible and is being developed with a goal of eventual operational use. Tensor methods can be applied to data sources beyond raw Zeek logs. The HACSAW environment provides access to enhanced versions of these logs along with a variety of other sensor data sources that have not yet been utilized. This paper describes three basic detectors, but others are possible including detectors for data exfiltration (possible over several protocols), lateral movement, and anomaly detection based on comparisons with historical decompositions. The HAGGLE platform and CFA framework are truly extensible supporting a variety of metadata sources and a limitless set of possible supporting analyses.

References

[1] M. Baskaran, T. Henretty, D. Bruns-Smith, J. Ezick, and R. Lethin, “Enhancing Network Visibility through Tensor Analysis,” in SC17 Innovating the Network for Data-Intensive Science (INDIS) Workshop, November 2017.

[2] M. Baskaran, T. Henretty, B. Pradelle, M. H. Langston, D. Bruns-Smith, J. Ezick, and R. Lethin, “Memory-efficient Parallel Tensor Decompositions,” in IEEE Conference on High Performance Extreme Computing (HPEC), September 2017.

[3] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, “Efficient and Scalable Computations with Sparse Tensors,” in IEEE High Performance Extreme Computing Conference, September 2012.

[4] M. Baskaran, B. Meister, and R. Lethin, “Low-overhead Load-balanced Scheduling for Sparse Tensor Computations,” in IEEE High Performance Extreme Computing Conference, September 2014.

[5] M. Baskaran, B. Meister, and R. Lethin, “Parallelizing and Optimizing Sparse Tensor Computations,” in Proceedings of the 28th ACM International Conference on Supercomputing, (ICS ’14), 2014, pp. 179–179.

[6] A. Gudibanda, T. Henretty, M. Baskaran, J. Ezick, and R. Lethin, "All-at-once Decomposition of Coupled Billion-scale Tensors in Apache Spark," in IEEE Conference on High Performance Extreme Computing (HPEC), September 2018.

[7] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling Tensor Analysis up by 100 Times – Algorithms and Discoveries," in KDD'12, August 2012.

[8] V. Castellana and M. Minutoli, “SHAD: The Scalable High-performance Algorithms and Data-structures Library,” in CCGRID 2018: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2018, pp. 442–451.

[9] L. Leonard and W. Glodek, "HACSAW: A Trusted Framework for Cyber Situational Awareness," in HoTSoS'18, April 2018.

[10] https://www.reservoir.com/research/tech/tensor-analysis/

[11] V. Castellana, M. Drocco, J. Feo, et. al., “A Parallel Graph Environment for Real-World Data Analytics Workflows,” in proceedings DATE 2019. Florence Italy, March 2019.

[12] https://www.zeek.org

[13] E. C. Chi and T. G. Kolda, “On Tensors, Sparsity, and Non-negative Factorizations,” arXiv:1304.4964 [math.NA], December 2011.

[14] M. Baskaran, T. Henretty, and J. Ezick, “Fast and Scalable Distributed Tensor Decompositions,” in IEEE Conference on High Performance Extreme Computing (HPEC), September 2019.

[15] K. Maruhashi, F. Guo, and C. Faloutsos, “Multiaspectforensics: Pattern Mining on Large-scale Heterogeneous Networks with Tensor Analysis,” in: International Conference on Advances in Social Networks Analysis and Mining, Kaohsiung, Taiwan, 2011.

[16] H.-H. Mao, W. Chung-Jung, E. E. Papalexakis, K.-C. L. Christos Faloutsos, and T.-C. Kao, “Malspot: Multi2 Malicious Network Behavior Patterns Analysis,” in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Tainan, Taiwan, 2014.

[17] C. Lorenzen, R. Agrawal, and J. King, "Determining Viability of Deep Learning on Cybersecurity Log Analytics," in 2018 IEEE International Conference on Big Data (Big Data), December 2018.

[18] https://chronicle.security/products/backstory/