Agenda

- Problem Description
- Background
- Methodology
- Results
- Future Work

Problem Description

Problem

- HPC systems have huge computing power, which can be misused, e.g. for:
  - Cryptocurrency miners
  - Password crackers
  - Other illicit workloads
- Misuse takes many forms:
  - Outsider hacking attacks
  - Account takeover
  - Account misuse by legitimate users

Let’s call all the things a supercomputer shouldn’t be used for “illicit workloads”

Real World Examples

US GOVERNMENT BANS PROFESSOR FOR MINING BITCOIN WITH A SUPERCOMPUTER

BRIAN COHEN • JUN 5, 2014

HPC wire

Search... Go

February 12, 2018

The Harvard Crimson

Harvard Research Computing Resources Misused for 'Dogecoin' Mining Operation

By Theodore R. Delwiche, Crimson Staff Writer

February 20, 2014

HPC wire

Search... Go

May 18, 2020

This week, a number of European supercomputers discovered intrusive malware hosted on their systems. Now, in the midst of a massive supercomputing research effort to tackle COVID-19, many enlisted systems have shut down or restricted access while they investigate and remove the malware.

The attacks may have been perpetrated in order to mine cryptocurrency; investigations are ongoing.

Problem

- Diverse paths to misuse (and diverse defenses), but the core of misuse is always the workload
- Examining workloads before they run is hard (halting problem)
- Examining workload after it runs is not useful
- Cannot impact performance of running system

Background

Terminology

- Illicit workload – any workload disallowed by policy or law
- Classification – the process of using machine learning to sort input into output buckets by type
- In-band data – data collected using a compute node’s operating system
- Out-of-band data – data collected without work by a compute node’s operating system

Illicit might include cryptocurrency miner or the use of classified software on non-classified systems

Prior work

- Use of in-band metric data relatively well explored
  - Peisert, “Fingerprinting Communication…“
  - Whalen et al. "Multiclass Classification…“
  - Whalen et al. "Network-theoretic Classification…“
  - DiMasi et al. "Identifying HPC codes…“
  - Ates et al. "Taxonomist: Application …“
- Other out-of-band methods explored
  - Combs et al. "Power Signatures…“

Peisert describes a system to classify workloads based on POSIX and MPI calls, gathered using a shim on NERSC systems

Whalen (the first), applies machine learning and shows it can operate on data similar to that from Peisert

Whalen (the second) uses an approach that requires understanding the data and it able to apply network theory in order to perform classification

Ates steps away from the MPI and POSIX monitoring and to more general hardwar emetrics collected by LDMS. It then shows that machine learning works on this LDMS data

Finally, Combs performs a physical modification of the system, to be able to read the power draw at a high frequency. They then perform machine learning on this data and are able to successfully determine the running workload

Problem (redux)

- All of these require system modifications
- In-band requires adding software, potential performance impacts and increase in complexity
- Out-of-band requires physical modifications to hardware

Overall Goal

Build a system which uses out-of-band data (collected without hardware modifications) to classify HPC workloads in real time

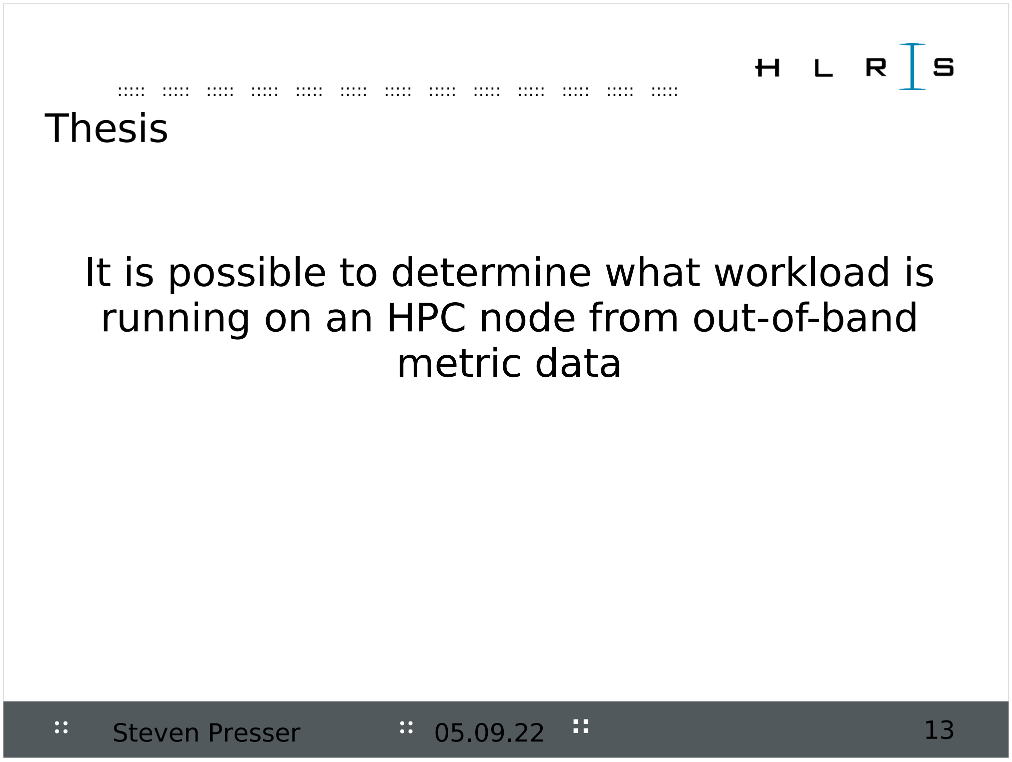
Given what we understand about the problem, what's the overall goal, beyond just this presentation and paper?

Overall Goal

- Out-of-band → No impact on system performance
- Real-time → Useful for detecting and reacting to illicit workloads

Why those elements? We’ve identified two problems in existing solutions. Some colelct collect data in-band and thus impact system performance. The corallry here is that some use out of band requiring hardware modifications, which is generally off the table for system support reasons.

The second problem is that many systems only can check after a job has finished running. Obviously, this isn’t very useful for detecting running illicit jobs – though it is useful for other purposes.



So here’s the thesis we’ll be testing:

Before jumping into the how...

What this is

- Initial work towards a larger goal
- An attempt to check the feasibility of an ML approach without the effort of collecting a novel data set

What this isn't

- Complete – there's much more to do!
- Proof positive that a particular approach works on a particular type of data – merely an indicator it might

Let's go over what this is and what this isn't.

So, it is an attempt to bite off a part of the big problem and work on it. Once we know that works, we can proceed to other parts. In that vein, this is an attempt to test the feasibility of an ML approach, without going through the significant effort of collecting a large and novel data set.

This isn't work that's complete – there's definitely more to do.

It also isn't absolute proof that anything that works will work on real out-of-band data. However, it is a strong indicator that it might

H L R I S

Taxonomist

- Dataset
- Existing results

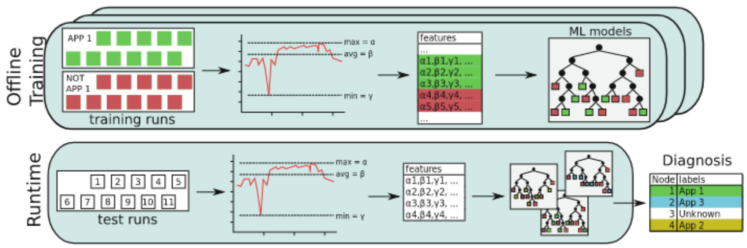


Fig. 3. Overview of Taxonomist.

I alluded to not collecting an entire novel data set of out-of-band data. So how will we test a thesis on out of band data?

Well, Ates made available a dataset called taxonomist, made up of in-band data from LDMS. Colelcted from Volta at Sandia. Generally speaking, out-of-band data overlaps with in-band data (or can be calculated from it). So we’ll use the taxonomist dataset as our base.

Conveniently, this also gives us a metric to compare against: how do our classification results compare to the in-band results from Ates? Ates took each job and calculated one input vector to their ML model from it – basically some cumulative metrics for the job.

Methodology

Outline of work

- Determine what data would be available out-of-band
- Determine intersection with Taxonomist dataset
- Reduce the data to the intersection
- Perform feature engineering
- Train model
- Evaluate model

Out-of-Band Data

- Currently two major ways to collect sensor data from systems:
  - IPMI (Intelligent Platform Management Interface)
  - RedFish
- Use RedFish as our basis

OoB data can be collected via the BMC – a tiny computer-within a computer used to control and measure nodes out of band. Working with the BMC has the advantage that the data often goes via a dedicated management network and won't cause congestion or jitter on the high speed network

RedFish is newer, (now) widely available, standardizes methodology of collecting data

Information available via RedFish

- Processor metrics
- Memory metrics
- Disk metrics
- Network metrics
- Environmental sensors
- Fan speeds

Redfish gives an awful lot of information about the system. It’s designed to do many things, among them give a lot of visibility into the inventory and state of a system. It includes everything from processor metrics to fan speeds, from disk I/O to network I/O and more.

Common between RedFish & Taxonomist

- Time in kernel mode
- Time in user mode
- Transmitted network frames
- Blocks allocated to cache
- Free memory
- Disk bytes read
- Power consumption

So the LDMS metrics in the taxonomist dataset of course have some overlap. Things like time in kernel and user mode, network frames transmitted, blocks in l3 cache, free memory, and some data on power consumption.

We’ll throw out everything in the taxonomist data set that isn’t this stuff, thus reducing it to things we know we can access out-of-band.

HLRS

Feature Engineering

- Largely follow Taxonomist methodology
- Parse data two ways:
  - Cumulative time series
  - Windowed time series

Armed with data, we move on to feature engineering

We largely follow the taxonomist methodology. That means for each metric collected we calculate the maximum, minimum, mean, standard deviation, skew, kurtosis,5th percentile, 25th percentile, 50th percentile, 75th percentile, and 95th percentile.

We use this to build two sequences of metrics. First, we use all the data cumulatively and calculate these metrics for all the data from the job. The last data point of each time series is equivalent to the data point for the job in Ates, other than being calculated for out-of-band data instead of in-band. We’ll therefore end up using this last point for comparison.

Second, we do a windowed version, with only the last 40 samples worth of data.

ML Models

- Trees
  - Random Forest
  - Extra Trees
  - Decision Tree
- SVC
  - LinearSVC
  - RBF Kernel
- Require 75% confidence, otherwise "unknown" instead

Experiments

- Taxonomist comparison
- Cumulative data
- Rolling window

For taxonomist comparison we use the last point of the cumulative data, as the is generated using identical methodology to how taxonomist extracts metrics for a time series

Results

Comparison to Taxonomist

Classifier	Data Type	Precision	Recall	F-Score
Random Forest	In-band	1.000	1.000	1.000
	Out-of-band	1.000	1.000	1.000
Extra Trees	In-band	1.000	1.000	1.000
	Out-of-band	1.000	1.000	1.000
Decision Tree	In-band	0.998	0.998	0.998
	Out-of-band	1.000	1.000	1.000
LinearSVC	In-band	0.999	0.999	0.999
	Out-of-band	0.987	0.904	0.942
SVC (RBF Kernel)	In-band	0.994	0.994	0.994
	Out-of-band	0.997	0.959	0.997

This shows that out-of-band data can be used for classification as accurately as in-band data, at least in some circumstances and using some methodology.

We’re going to drop SVC methods moving forward. Between poor performance on out of band data and exponetial training times, they’re not suitable for the next experiments

::::: ::::: ::::: ::::: ::::: ::::: ::::: ::::: ::::: ::::: ::::: ::::: :::::

HLRS

Cumulative Classification

Classifier	Precision	Recall	F-Score	Classifications/s
Random Forest	1.000	0.996	0.998	18,984
Extra Trees	1.000	0.999	0.999	20,336
Decision Tree	0.999	0.999	0.999	317,722

Next up is classification performance when run over every point in time, not just the end result. This allows us to simulate classification on an ongoing basis during the running of a job. Basically, if we were watching a job the entire time it ran and performing classification every time we got new metrics, how accurate would be be?

The answer turns out to be quite accurate. These are for cumulative classification – basically keeping the entire metric history to date and forming metrics based on that. In the real world, this could be memory and computation expensive, so...

Rolling Classification

Classifier	Precision	Recall	F-Score	Classifications/s
Random Forest	1.000	0.996	0.998	18,796
Extra Trees	1.000	0.999	0.999	18,058
Decision Tree	0.996	0.996	0.996	311,165

Steven Presser 05.09.22 27

The same experiments were re-run with a window. In this case, a 40-sample window. The results are just about as accurate. In theory, such a window should have a constant memory and computation footprint, making it much easier to capacity plan in the real world.

The results for both cumulative and rolling classification show high precison and recall, indicating that if we were continuously running classification using the generated ML models, it would perform pretty accurately.

Theoretical Cores for Classification

System Name	June 2022 Top 500	Nodes	Cores for Classification once per Second
Frontier	1	9,408	1
Fugaku	2	158,976	9
LUMI	3	2,560	1
Summit	4	4,356	1
Sierra	5	4,320	1
HAWK	27	5,632	1

Speaking of running classification when we get new metrics and of capacity planning – how expensive would it be to actually run classification? Do we need to buy clusters to perform this classification?

As it turns out, probably not. The slowest ML model ran 18,796 classifications per second. Assuming we collect metrics once per second from each node – a rather fast rate – for a significant portion of the top supercomputers, we still only need one core.

These numbers do exclude everything that isn’t the classification itself – calculating metrics, network, etc. They also assume exactly one out-of-band controller per node, which is not always the case.

Summary

- Precision/recall of classification on simulated out-of-band data as good as those in Ates
- High precision/recall for cumulative classification
- High precision/recall for rolling classification

So, what have we shown?

First, that performing machine learning on simulated out of band data produces classification as accurate as the in-band data. Or, put otherwise, out-of-band data is as good for figuring out what’s running as in-band data.

Second, that performing classification over cumulative metrics produces largely good results, This shows that we can start performing classification pretty early in a job and expect it to perform well.

And third, that rolling classification works with not much precision/recall loss. This means that is cumulative is too expensive in memeory or computation, we can fix the size to a window and get results that are almost as good.

Future Work

Future Work

- Re-run with real out-of-band data
- Experiment with other ML methods
- Gather data for and include “malicious” software
- Gather data from multiple HPC systems – how well do these methods/models translate from system to system?
- Build a true real-time detection system for illicit workloads!

Funding Acknowledgment

This work has been supported by the project InHPC-DE, which received funding from the Ministry of Education and Research (BMBF), Germany.

Questions?

Want to collaborate?

Email me: steven.presser@hlrs.de