Load Shedding in Network Monitoring Applications

Pere Barlet-Ros[1] Gianluca Iannaccone[2]
Josep Sanjuàs[1] Diego Amores[1] Josep Solé-Pareta[1]

[1]Technical University of Catalonia (UPC)
Barcelona, Spain

[2]Intel Research
Berkeley, CA

Intel Research Berkeley, July 26, 2007

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Outline

1. Introduction
   - Motivation
   - Case Study: Intel CoMo
2. Load Shedding
   - Prediction Method
   - System Overview
3. Evaluation and Operational Results
   - Performance Results
   - Accuracy Results

Motivation

- Building robust network monitoring applications is hard
  - Unpredictable nature of network traffic
  - Anomalous traffic, extreme data mixes, highly variable data rates
- Processing requirements have greatly increased in recent years
  - E.g., intrusion and anomaly detection

Motivation

- Building robust network monitoring applications is hard
  - Unpredictable nature of network traffic
  - Anomalous traffic, extreme data mixes, highly variable data rates
- Processing requirements have greatly increased in recent years
  - E.g., intrusion and anomaly detection

The problem

- Efficiently handling extreme overload situations
- Over-provisioning is not possible

Case Study: Intel CoMo

- CoMo (Continuous Monitoring)[1]
  - Open-source passive monitoring system
  - Fast implementation and deployment of monitoring applications
- Traffic queries are defined as *plug-in* modules written in C
  - Contain complex stateful computations

[1] `http://como.sourceforge.net`

Case Study: Intel CoMo

- CoMo (Continuous Monitoring)[1]
  - Open-source passive monitoring system
  - Fast implementation and deployment of monitoring applications
- Traffic queries are defined as *plug-in* modules written in C
  - Contain complex stateful computations

Traffic queries are black boxes

- Arbitrary computations and data structures
- Load shedding cannot use knowledge about the queries

[1] http://como.sourceforge.net

Load Shedding Approach

Main idea

1. Find correlation between **traffic features** and CPU usage
   - Features are **query agnostic** with **deterministic worst case** cost
2. Leverage correlation to predict CPU load
3. Use prediction to guide the load shedding procedure

Load Shedding Approach

Main idea

1. Find correlation between **traffic features** and CPU usage
   - Features are **query agnostic** with **deterministic worst case** cost
2. Leverage correlation to predict CPU load
3. Use prediction to guide the load shedding procedure

Novelty: No *a priori* knowledge of the queries is needed

- Preserves high degree of flexibility
- Increases possible applications and network scenarios

Traffic Features vs CPU Usage

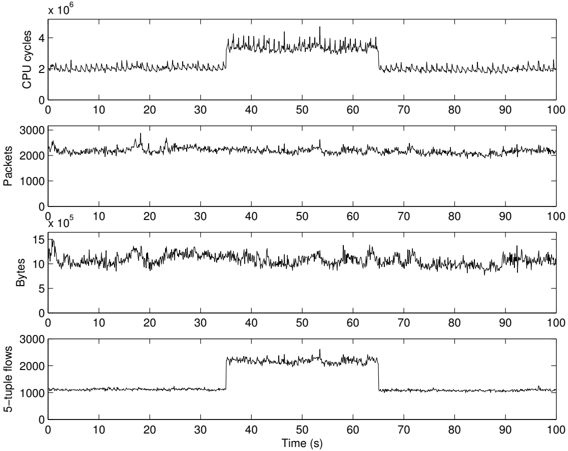


Figure: CPU usage compared to the number of packets, bytes and flows

System Overview

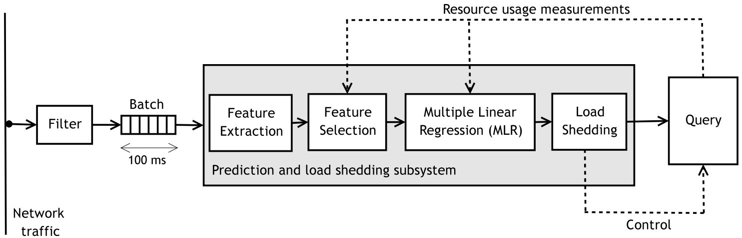


Figure: Prediction and Load Shedding Subsystem

Load Shedding Performance

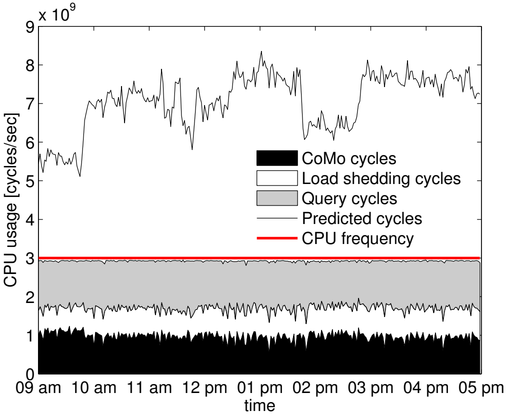


Figure: Stacked CPU usage (Predictive Load Shedding)

Load Shedding Performance

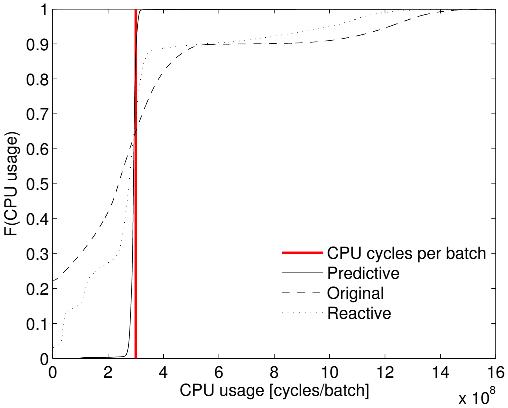


Figure: CDF of the CPU usage per batch

Accuracy Results

- Queries estimate their unsampled output by multiplying their results by the inverse of the sampling rate
- Errors in the query results (*mean* ± *stdev*)

Query	***original***	***reactive***	***predictive***
application (pkts)	55.38% ±11.80	10.61% ±7.78	1.03% ±0.65
application (bytes)	**55.39% ±11.80**	**11.90% ±8.22**	**1.17% ±0.76**
flows	38.48% ±902.13	12.46% ±7.28	2.88% ±3.34
high-watermark	8.68% ±8.13	8.94% ±9.46	2.19% ±2.30
link-count (pkts)	55.03% ±11.45	9.71% ±8.41	0.54% ±0.50
link-count (bytes)	55.06% ±11.45	10.24% ±8.39	0.66% ±0.60
top destinations	21.63 ±31.94	41.86 ±44.64	1.41 ±3.32

Ongoing and Future Work

- Ongoing Work
  - Query utility functions
  - Custom load shedding
  - Fairness of service with non-cooperative users
  - Scheduling CPU access vs. packet stream
- Future Work
  - Distributed load shedding
  - Other system resources (memory, disk bandwidth, storage space)

Availability

- The source code of our load shedding prototype is publicly available at http://loadshedding.ccaba.upc.edu
- The CoMo monitoring system is available at http://como.sourceforge.net

Acknowledgments

- This work was funded by a University Research Grant awarded by the Intel Research Council and the Spanish Ministry of Education under contract TEC2005-08051-C03-01
- Authors would also like to thank the Supercomputing Center of Catalonia (CESCA) for giving them access the Catalan RREN

Work Hypothesis

Our thesis

- Cost of mantaining data structures needed to execute a query can be modeled looking at a set of traffic features

Empirical observation

- Different overhead when performing basic operations on the state while processing incoming traffic
  - E.g., creating or updating entries, looking for a valid match, etc.
- Cost of a query is mostly dominated by the overhead of some of these operations

Work Hypothesis

Our thesis

- Cost of mantaining data structures needed to execute a query can be modeled looking at a set of traffic features

Empirical observation

- Different overhead when performing basic operations on the state while processing incoming traffic
  - E.g., creating or updating entries, looking for a valid match, etc.
- Cost of a query is mostly dominated by the overhead of some of these operations

Our method

Models queries' cost by considering the **right set** of traffic features

Traffic Features vs CPU Usage

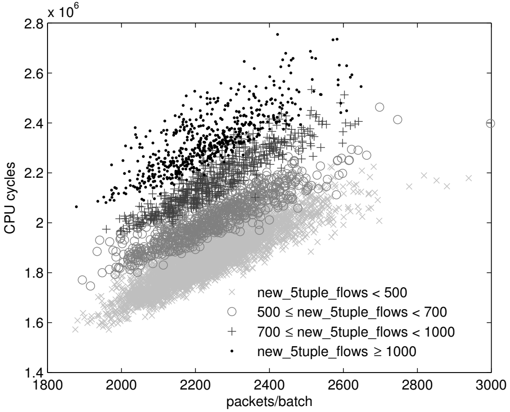


Figure: CPU usage versus the number of packets and flows

Multiple Linear Regression (MLR)

Linear Regression Model

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi} + \varepsilon_i, \qquad i = 1, 2, \ldots, n.$$

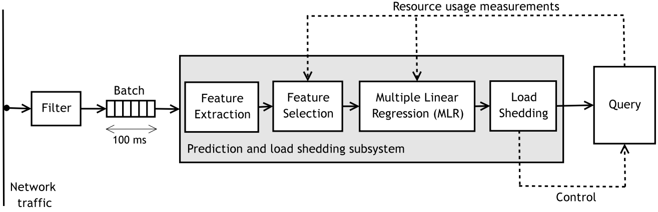
- Y_i = n observations of the response variable (measured cycles)
- X_{ji} = n observations of the p predictors (traffic features)
- β_j = p regression coefficients (unknown parameters to estimate)
- ε_i = n residuals (OLS minimizes SSE)

Feature Selection

- Variant of the Fast Correlation-Based Filter[2] (FCBF)
- Removes **irrelevant** and **redundant** predictors
- Reduces significantly the cost of the MLR

[2] L. Yu and H. Liu. Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. In *Proc. of ICML*, 2003.

System Overview



Prediction and Load Shedding subsystem

1. Each 100*ms* of traffic is grouped into a *batch* of packets
2. The traffic features are efficiently extracted from the batch (multi-resolution bitmaps)
3. The most relevant features are selected (using FCBF) to be used by the MLR
4. MLR predicts the CPU cycles required by the query to run
5. Load shedding is performed to discard a portion of the batch
6. CPU usage is measured (using TSC) and fed back to the prediction system

Load Shedding

When to shed load

- When the prediction exceeds the available cycles
- $avail\_cycles = (0.1 \times CPU\ frequency) - overhead$
  - Corrected according to prediction error and buffer space
  - Overhead is measured using the time-stamp counter (TSC)

How and where to shed load

- Packet and Flow sampling (hash based)
- The same sampling rate is applied to all queries

How much load to shed

- Maximum sampling rate that keeps CPU usage $< avail\_cycles$
- $srate = \frac{avail\_cycles}{pred\_cycles}$

Load Shedding

When to shed load

- When the prediction exceeds the available cycles
- $avail\_cycles = (0.1 \times CPU\ frequency) - overhead$
  - Corrected according to prediction error and buffer space
  - Overhead is measured using the time-stamp counter (TSC)

How and where to shed load

- Packet and Flow sampling (hash based)
- The same sampling rate is applied to all queries

How much load to shed

- Maximum sampling rate that keeps CPU usage $< avail\_cycles$
- $srate = \frac{avail\_cycles}{pred\_cycles}$

Load Shedding

When to shed load

- When the prediction exceeds the available cycles
- $avail\_cycles = (0.1 \times CPU\ frequency) - overhead$
  - Corrected according to prediction error and buffer space
  - Overhead is measured using the time-stamp counter (TSC)

How and where to shed load

- Packet and Flow sampling (hash based)
- The same sampling rate is applied to all queries

How much load to shed

- Maximum sampling rate that keeps CPU usage $< avail\_cycles$
- $srate = \frac{avail\_cycles}{pred\_cycles}$

Load Shedding Algorithm

Load shedding algorithm (simplified version)

pred_cycles = 0;
foreach q_i *in Q* **do**
  f_i = `feature_extraction(`b_i`);`
  s_i = `feature_selection(`f_i, h_i`);`
  pred_cycles += `mlr(`f_i, s_i, h_i`);`

if $avail\_cycles < pred\_cycles \times (1 + \widehat{error})$ **then**
  foreach q_i *in Q* **do**
    b_i = `sampling(`$b_i, q_i, srate$`);`
    f_i = `feature_extraction(`b_i`);`

foreach q_i *in Q* **do**
  $query\_cycles_i$ = `run_query(`$b_i, q_i, srate$`);`
  h_i = `update_mlr_history(`$h_i, f_i, query\_cycles_i$`);`

Testbed Scenario

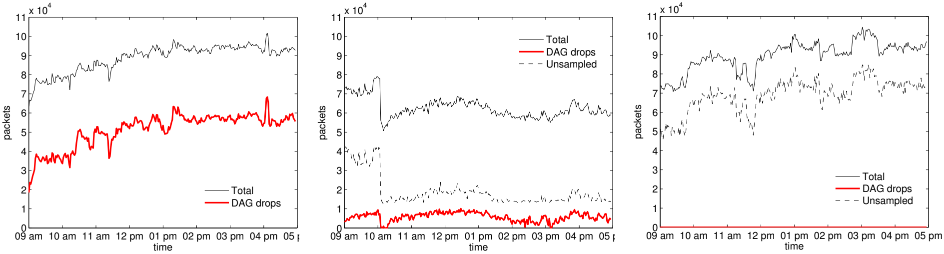
- Equipment and network scenario
  - 2 × Intel® Pentium™ 4 running at 3 *GHz*
  - 2 × Endace® DAG 4.3GE cards
  - 1 × Gbps link connecting Catalan RREN to Spanish NREN
- Executions

Execution	Date	Time	Link load (Mbps) mean/max/min
predictive	24/Oct/06	9am:5pm	750.4/973.6/129.0
original	25/Oct/06	9am:5pm	719.9/967.5/218.0
reactive	05/Dec/06	9am:5pm	403.3/771.6/131.0

- Queries (from the standard distribution of CoMo)

Name	Description
application	Port-based application classification
counter	Traffic load in packets and bytes
flows	Per-flow counters
high-watermark	High watermark of link utilization
pattern search	Finds sequences of bytes in the payload
top destinations	List of the top-10 destination IPs
trace	Full-payload collection

Packet Loss



(a) Original CoMo (b) Reactive Load Shedding (c) Predictive Load Shedding

Figure: Link load and packet drops

Related Work

Network Monitoring Systems

- Only consider a pre-defined set of metrics
- Filtering, aggregation, sampling, etc.

Data Stream Management Systems

- Define a declarative query language (small set of operators)
- Operators' resource usage is assumed to be known
- Selectively discard tuples, compute summaries, etc.

Related Work

Network Monitoring Systems

- Only consider a pre-defined set of metrics
- Filtering, aggregation, sampling, etc.

Data Stream Management Systems

- Define a declarative query language (small set of operators)
- Operators' resource usage is assumed to be known
- Selectively discard tuples, compute summaries, etc.

Limitations

- Restrict the type of metrics and possible uses
- Assume explicit knowledge of operators' cost and selectivity

Conclusions and Future Work

- Effective load shedding methods are now a basic requirement
  - Rapidly increasing data rates, number of users and complexity of analysis methods
- Load shedding operates without knowledge of the traffic queries
  - Quickly adapts to overload situations by gracefully degrading accuracy via packet and flow sampling
- Operational results in a research ISP network show that:
  - The system is robust to severe overload
  - The impact on the accuracy of the results is minimized
- Limitations and Future work
  - Load shedding methods for queries non robust against sampling
  - Load shedding strategies to maximize the overall system utility
  - Other system resources (memory, disk bandwidth, storage space)