Handling Big Streaming Data with DILoS

Alexandros Labrinidis

Advanced Data Management Technologies Lab
Department of Computer Science
University of Pittsburgh

University of Waterloo – May 21, 2014
You know Big Data is an important problem if...

- It is featured on the cover of Nature and the Economist!
You know Big Data is an *even more* important problem if...

- It has a Dilbert cartoon!
What is Big Data?

Definition #1:
• Big data is like teenage sex:
  o everyone talks about it,
  o nobody really knows how to do it,
  o everyone thinks everyone else is doing it,
  o so everyone claims they are doing it...

Definition #2:
• Anything that Won't Fit in Excel!

Definition #3:
• Using the Vs
The three Vs

- **Volume** - size does matter!
- **Velocity** - data at speed, i.e., the data “fire-hose”
- **Variety** - heterogeneity is the rule
Five more Vs

- **Variability** - rapid change of data characteristics over time
- **Veracity** - ability to handle uncertainty, inconsistency, etc
- **Visibility** – protect privacy and provide security
- **Value** – usefulness & ability to find the right-needle in the stack
- **Voracity** - strong appetite for data!
Enter Moore's Law

Microprocessor Transistor Counts 1971-2011 & Moore's Law

Log Scale!

Transistor count

- Curve shows transistor count doubling every two years

Date of introduction


[ Wikipedia Image ]
Storage capacity increase

- HDD Capacity (GB)

[ Wikipedia Data ]
But

- **Human processing capacity** remains roughly the same!
We refer to this as the:

Big Data – Same Humans Problem
Roadmap

% of audience asleep

# of slides

- Big Data Intro ✔
- ADMT Lab Intro
- AQSIOS DSMS
- ALoMA
- DILoS
- Conclusions
About the ADMT Lab

• Directed by
  • Panos K. Chrysanthis
  • Alexandros Labrinidis

• Established in 1995

• 4+2 PhD students, 2 MS students, 6 REUs

• User-centric data management for network-centric applications
Entire Data Lifecycle

Data Acquisition

Web Data Management

Data Flow

Data Stream Processing

Data Dissemination
AstroShelf

- Understanding the Universe through scalable navigation of a galaxy of annotations

- Astronomy data from multiple sources (images & catalogs)

- Support collaboration of:
  - people (view-based, declarative annotations)
  - software / data (web services)
  - resources (utilizing local and remote storage)

- CONFLuEnCE prototype: continuous workflows [Sigmod 2011 & 2012]
• **User-centric features:**

  -1 0.2 m3 m4
  0.2 0.8 0.8
  m2 m1 m5

  “I like drama movies a bit more than horror movies, Intensity of preference 0.2”

- Unified model for user preferences
  - combine quantitative & qualitative user preferences into a single graph model to guide query result personalization

- Protecting privacy in distributed query processing
  - declarative preferences allow users to balance the tradeoff between privacy and performance

```
SELECT * FROM Plants, Supplies, Polluted_H2O
WHERE Supplies.type = "solvent"
AND Supplies.name = Polluted_H2O.pollutant
AND Polluted_H2O.location = Plants.location
AND Plant.id = Supplies.plant_id
PREFERRING $l = Querier HOLDS OVER <*,{(pollutant)},$l>
CASCADE LESSTHAN(runtime, 120)
AND $l = Querier HOLDS OVER <join,*,$l>;
```
Efficiently Utilizing Resource in a Data Stream Management System

CPU time sharing:
- Which operator to execute now?
- And for how long?

What if the system is overloaded?
- Shed data to meet the near-real-time requirement

Which query plans are the best?

Multiple classes of CQs
- Each class has a different priority
• Prototype Data Stream Management Systems
  - **Aggregate Continuous Query optimizer**
    - WeaveShare and TriWeave
      [Shenoda et al., CIKM’11 and ICDE’12]
      - Optimized processing to eliminate redundant computation
  - **Continuous Query Schedulers**
    - HR, HNR [Sharaf et al., VLDB’06 and TODS’08]
      - Average vs Max Response Time
      - Average vs Max Slowdown
    - CQC and ABD [Al Moakar et al., DMSN’09 and SMDB’12]
      - Priority Classes
      - Single-, Dual-, Multi-core, Cloud
AQSIOs (cont.)

- Load shedder and scheduler-load shedder synergy
  - **SEaMLeSS** [Pham et al., SMDB’13]
    - SEIf Managing Load Shedding for data Stream management systems
  - **DIloS** [Pham et al., SMDB’11]
    - Seamless integration of priority-based scheduler and load shedder
    - Consistently honor worst-case delay target with differentiated classes of service
    - Exploit system capacity better
Roadmap

% of audience asleep vs. # of slides

- Big Data Intro
- ADMT Lab Intro
- AQSIOS DSMS
- ALoMA
- DILoS
- Conclusions
System Model & Metrics

• **Multiple priority classes** of CQs
  o Priorities have been quantified into numbers
    • Higher value means higher priority

• Two requirements under *overload state*:
  1. **Guarantee worst-case Quality of service (QoS)**
     • Worst-case QoS = worst-case response time = delay target
     • Each class can require a different worst-case QoS
     • Supported by load manager (load shedder)

  2. **Maximize Quality of Data (QoD) with priority consideration**
     • QoD = 100% - data loss due to shedding
     • Need to consider priorities of CQ classes
     • **Involve both scheduler and load manager** - Why?
State-of-the-art

• Previous works consider either…

  o Priority-based scheduling
    • CQ’s priority (through QoS function, deadline): e.g., [Carney et al., VLDB’03], [Wei et al., ISORC’06]
    • Class’ priority: [Al Moakar et al., DMSN’09, SMDB’12]

  o Or priority-based load shedding
    • CQ’s loss-tolerance functions [Tatbul et al., VLDB’03]

Now we need both of them to work together …
Motivation

• Two CQs $Q_1$ and $Q_2$
  o The same cost
  o $Q_1$’s priority is twice as high as $Q_2$’s

Input rate

Scheduler:

Load manager:

$\rightarrow Q_2$ is still overloaded
$\rightarrow Q_1$ suffers from unnecessary shedding
$\rightarrow$ System capacity is not fully used
Motivation

• Making the load manager aware of the scheduler’s policy?
  
  o **Load manager**: I should know that the scheduler can process up to 10 tuples of $Q_1$ and 5 tuples of $Q_2$ and...
  
  o **Scheduler**: well, all I can tell you is in this cycle I am giving $Q_1$ x% of time to execute and $Q_2$ y% and..., also many things out of my control
    • Context switching time
    • Background jobs that share the CPU resource
    • The actual query load
  
  o **Load manager**: 😞
Our Hypothesis

• By exploiting the **synergy** between the scheduler and the load shedder we can

  o Support CQ’s priority consistently

  o Improve the utilization of CPU resource
Our solution: DILoS framework

- Capacity usage
- Demand
- Budget distribution
- Supply

2-level scheduler (e.g., [Al Moakar SMDB’09])

Per-class load manager

- Global scheduler
- Local sched. 1
  - Class 1
    - Load manager 1
- Local sched. 2
  - Class 2
    - Load manager 2
- Local sched. k
  - Class k
    - Load manager k

budget = Σsupplyᵢ
Benefit of our proposed DILoS framework

• The load manager works in concert with the scheduler in honoring CQs’ priority
  o The load manager does not need to have its own priority-based policy
    • Controls the load in each class as if it is a virtual system
    • Follows exactly the priority enforcement of the scheduler

• Load manager’s feedback improves scheduler’s decision
  o Better exploits system capacity
Roadmap

% of audience asleep vs # of slides

- Big Data Intro
- ADMT Lab Intro
- AQSIOS DSMS
- ALoMA: Adaptive Load Manager
- DILoS
- Conclusions
Load manager for DILoS

• Each class load manager needs to decide "when and how much load to shed"

  o Estimate the load of each class
    • [Tatbul et al., 2003], based on input rates, operator’s cost and selectivities

  o Estimate the system capacity each class actually has
    • ???
“When and how much”- related definitions

• Incoming load $L$
  o The amount of time needed to process all the tuples coming in per time unit (say, a second)

• System capacity $L_C$:
  o The fraction of each time unit the system can spend on processing the incoming tuples
  o Approximated by a headroom factor $H$ in [0-1]

• Overload:
  o when $L > L_C$
“when and how much” state-of-the-art

- **Aurora [Tatbul et al., 2003]**
  - Excess load = $L - L_C$
  - No feedback loop, cannot honor delay target

- **CTRL [Tu et al., 2006]**
  - Based on number of queued tuples to adjust shedding decisions
  - Honors delay target, outperforms Aurora

- **Both require manually tuned headroom factor $H$ to estimate the system capacity!**
  - Offline, manual tuning of $H$ is impractical
  - Clearly not applicable in this context of per-class load manager!
Our Proposal: ALoMa – Adaptive Load Manager

- **Starts with some reasonable value of H, and adjusts it accordingly**

- **Has two modules:**
  - *Statistics–based load monitor:* estimates the system load based on input rate, operators’ costs and selectivities
  - *Response time monitor:* monitors the level and moving trend of the actual response time to infer about the system load status
ALoMa- Headroom Factor Adjustment

• The two modules disagree: adjust $H$
  
  o The load monitor says “overloaded” but the response time monitor says “not overloaded”:
    
    • **Increase** $H$ so that $L_C$ is increased towards $L$
  
  o The load monitor says “not overloaded” but the response time monitor says “overloaded”
    
    • **Decrease** $H$ so that $L_C$ is reduced towards $L$

• The two modules agree: excess load = $L - L_C$
ALoMa – Headroom Factor Adjustment

• We use heuristic in the adjustment of H (or $L_C$)
  o Accommodating system fluctuation and the inherent lag of the statistics

\[
L_{C_{new}} = L_C \pm \frac{\log_2(z + 1)}{z} |L - L_C|
\]

where $z = \begin{cases} 
\frac{|L - L_C|}{L_C} \cdot 0.100 & \text{if } \frac{|L - L_C|}{L_C} \cdot 0.100 \geq 1 \\
1 & \text{otherwise}
\end{cases}$

o Principle: bigger the difference, smaller the % of change but bigger in absolute value of change
Effect of environment changes on CTRL [Tu et al.] and adaptation of ALoMa. Total data loss for ALoMa and CTRL is 62.98% and 62.69%, respectively.
We showed how ALoMa can automatically recognize the system capacity spent on query processing.

ALoMa’s other important advantages over the state-of-the-art:

<table>
<thead>
<tr>
<th>Ideal properties</th>
<th>ALoMa</th>
<th>CTRL</th>
<th>Aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aware of delay target</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Auto-adjusting of H</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicable to all query networks</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Independent of scheduler</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Roadmap

% of audience asleep vs. # of slides

- Big Data Intro
- ADMT Lab Intro
- AQSIOS DSMS
- DILoS
- ALoMA: Adaptive Load Manager
- Conclusions

36% of audience asleep.
Back to DILoS Framework

Global scheduler

Capacity distribution policy

Class’ capacity usage

ALoMa 1
Local (operator) scheduler 1
Class 1

... ...

ALoMa k
Local (operator) Scheduler k
Class k

... ...

ALoMa N
Local (operator) scheduler N
Class N
Scheduling Policy

• A concrete policy implemented:
  o A class with priority $P_k$ is guaranteed a share of $\frac{P_k}{\sum_{i=1}^{N} P_i}$ of total system processing capacity if needed.
    • Adopted from CQC [Al Moakar et al., 2009]
  o Redundant capacity from a class is distributed to other classes in need with “highest priority first”

• Different policies can be plugged in, for example:
  o Absolute priority for higher-priority class:
    • Higher class can use as much of the available capacity as needed
  o Relative priority with workload consideration
    • Higher class receives better QoD regardless of its workload
Inter-class Sharing

- **Congestion** can happen when a higher-priority class share a query segment with a lower-priority one under class-based scheduling.

- The shared segment receives the higher-priority as it should.

- However, the higher-priority class is blocked waiting for the lower priority one to consume the intermediate result.

  ➔ DILoS naturally provides a solution, enabling inter-class operator sharing.

Claim: As long as the load of the lower-priority class is controlled to its capacity, congestion will not happen.
Experiments

Experimental Settings

• AQSIOS DSMS prototype

• Three classes 1, 2, 3 of priorities 6, 3, 1; 6 is the highest
• All classes have the same workload of 11 queries
• Worst-case QoS of class 1, 2, 3 is 300, 400, 500 ms

• Input rate:
  o Constant, step changes, and real input trace for class 1
  o Constant input rate for class 2 and 3, at a level that would overload the classes within its assigned capacity.
## Result with Constant Input Rate

<table>
<thead>
<tr>
<th></th>
<th>Average response time (ms)</th>
<th>Average data loss (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class 1</td>
<td>Class 2</td>
</tr>
<tr>
<td>No load manager</td>
<td>3.40</td>
<td>3.53</td>
</tr>
<tr>
<td>Common load manager</td>
<td>3.00</td>
<td>3.13</td>
</tr>
<tr>
<td>Per-class load manager</td>
<td>3.55</td>
<td>3.75</td>
</tr>
<tr>
<td>DILoS</td>
<td>4.28</td>
<td>4.38</td>
</tr>
</tbody>
</table>
Understand the Benefit of the Synergy

Implicit redistribution observed without explicit synergy

Data loss:
- Class 1: 0%
- Class 2: 0%
- Class 3: 35.9%

Higher than 0.1!

Explicit synergy and redistribution

Data loss:
- Class 1: 0%
- Class 2: 0%
- Class 3: 0%

⇒ Better capacity usage by exploiting batch processing!
Enabling inter-class sharing

Class 1 shares a query segment with class 3 under a class-based scheduling policy (CQC [Al Moakar et al., 2011]) (constant input rate)

Congestion

We solved it with DILoS
Result with Step Changes in Class 1’s Input Rate

Figure 12: Response times under DILoS, with step changes in input rate of class 1

Figure 13: Shedding and estimated headroom factors under DILoS, with step changes in input rate of class 1 to the other two classes, enabling them to shed less. However, as soon as the load of class 1 increases (e.g., at the 100th second), DILoS gives back to this class all or part of its original capacity so that its performance, as specified by its class priority, is preserved.

3.3.3 Extensibility

As a framework with two-level integrated scheduling and load management, DILoS enables easy incorporation of different scheduling and load shedding schemes at both the global and local level. At the global level, different capacity allocation and redistribution policies can be adopted once the scheduler obtains the report from the load manager regarding the capacity usage of each class. At the second, local level, different load shedders and operator schedulers can be used. We discuss in this section these possibilities.

Different Capacity allocation and redistribution policies

The beauty of our proposed scheduler and load-shedder synergy is that it is not limited to a single policy. For exposition, we use in this paper the extended CQC policy that is sound in some context,
Result with Step Changes in Class 1’s Input Rate

Figure 12: Response times under DILoS, with step changes in input rate of class 1

Figure 13: Shedding and estimated headroom factors under DILoS, with step changes in input rate of class 1

class 1 to the other two classes, enabling them to shed less. However, as soon as the load of class 1 increases (e.g., at the 100th second), DILoS gives back to this class all or part of its original capacity so that its performance, as specified by its class priority, is preserved.

3.3.3 Extensibility

As a framework with two-level integrated scheduling and load managing, DILoS enables the incorporation of different scheduling and load shedding schemes at both the global and local level. At the global level, different capacity allocation and redistribution policies can be adopted once the scheduler obtains the report from the load manager regarding the capacity usage of each class. At the second, local level, different load shedders and operator schedulers can be used. We discuss in this section these possibilities.

Different Capacity allocation and redistribution policies

The beauty of our proposed scheduler and load-shedder synergy is that it is not limited to a single policy. For exposition, we use in this paper the extended CQC policy that is sound in some context.
In this set of experiments we replace the synthetic input increases (e.g., from the bottom of this figure we repeat the real input rate pattern for each of the three classes, we also plot the headroom DILoS with inter-class sharing in Fig. 20. In order to very high peaks. periods when the rate keeps increasing with sudden, the normal fluctuations in the system. small and often not observed because it is obscured by time windows. The additional data loss, however, is very high compared to the normal capacity redistribution (much more data is saved: 3.28% vs 7.49% increase in the shedding rate of class 1 (0.45%) is much more significant). This occurs when the input rate fluctuates considerably (after the huge increase in the input rate caused overloading, and since class 1 passed its excess capacity to the others, its lag of the statistics-based decision. More specifically, load shedding, before the scheduler could subsequently, load shedding, before the scheduler could recognize and correct the situation.

The results also show the benefit of sharing in saving redundant capacity from other classes and allowing the class' capacity (while still allowing the class to use its policy that includes a limit on the shared usage of excess capacity the class can give to the other classes). In such a case, the lag of the statistics-based decision causes small additional shedding in some of the classes have very light load, as mentioned. The implicit redistribution of the system capacity when some of the classes have very light load, as mentioned, if a class is highly critical and such a trade-off is tolerated, one can develop a capacity redistribution policy that includes a limit on the shared usage of excess capacity, including those used by DILoS to enforce explicit sharing does not affect the QoS and QoD of the higher priority class.

The real input is the trace of TCP packages to and from The Berkeley Lab (http://ita.ee.lbl.gov/html/contrib/LBL-PKT.html)
Result with Real Input Rate for Class 1’s

<table>
<thead>
<tr>
<th></th>
<th>Average response time (ms)</th>
<th>Average data loss (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class 1</td>
<td>Class 2</td>
</tr>
<tr>
<td>No synergy (&amp; no sharing)</td>
<td>22.31</td>
<td>68.23</td>
</tr>
<tr>
<td>DILoS without sharing</td>
<td>25.69</td>
<td>76.86</td>
</tr>
<tr>
<td>DILoS with sharing</td>
<td>25.03</td>
<td>70.29</td>
</tr>
</tbody>
</table>
Roadmap

% of audience asleep

# of slides

Big Data Intro ✔

ADMT Lab Intro ✔

AQSIOs DSMS ✔

ALoMA ✔

DILoS ✔

Conclusions

48% of audience asleep
Conclusions

• Advantages of DILoS:
  o Seamless integration:
    • The load manager *detects* and *follows exactly* the current priority enforcement of the global scheduler
  o Global scheduling decision improved
    • *Explicitly control the distribution* of available capacity
    • *Exploit batch processing* to increase capacity utilization
    • Enable inter-class sharing to maximize the chance for query optimization
  o Different priority policies can be plugged in

• Future works:
  o Synergy with priority-based memory management
  o Consider advanced architecture (multi-core, cloud)
A (Big) Team Effort

Faculty

• Panos Chrysanthis
• Alexandros Labrinidis
• Adam Lee
• Kirk Pruhs

Students

• Lory Al Moakar
• Di Bao
• Nick Farnan
• Roxana Gheorghiu
• Shenoda Guirguis
• Qinglan Li
• Panickos Neophytou
• Thao Pham
• Mohamed Sharaf
• Matt Schroeder
• Nikhil Venkatesh

FUNDING (DILoS)

• NSF IIS-0534531
• NSF CAREER IIS-0746696
• EMC/Greenplum
• Andrew Mellon Predoctoral Fellowship

http://db.cs.pitt.edu