Big data for big data: Machine-learning models of Hadoop cluster behavior

Sean Suchter and Shekhar Gupta
Who are we?

Sean Suchter = CTO & co-founder, Pepperdata. Previously 15 years in web search (Inktomi, Yahoo!, Bing)

Dr Shekhar Gupta = Software Engineer, Pepperdata. PhD in Optimizing Resource Utilization in Distributed Systems
What is in this talk?

- **Swapping & Thrashing**
  - What are they, in general
  - How do they manifest in Hadoop
- **Good swapping vs Bad swapping (aka Thrashing)**
- **How can thrashing be avoided (basic approaches)**
- **How can thrashing can be avoided (dynamic, supervised learning approach)**
  - Inputs used
  - Algorithm
  - Learned parameters
  - Action taken
- **Results & wrap-up**
What is swapping?

When the processes running on server instance attempt to allocate more memory than your system has available, the kernel begins to *swap* memory pages to and from the disk.
Swapping in YARN (Hadoop)

N is determined based on configured memory and number of cores per node
Swapping in YARN (Hadoop)

Example container:

Scheduler

N containers

C_1

C_2

C_N

Physical Memory Used RSS Bytes by Container
Good Swapping

Common case:

- Memory of some processes is **not actively being used**
  - It’s ok for this memory to be swapped out to disk

Another case:

- **Transient** overload of memory
  - E.g. fork/exec of a program that might go away (example: cron job)
Bad Swapping (aka “thrashing”)  

Cause:

- Too much active memory need compared to physical RAM
- As processes try to use CPU, their pages are read and other are written, constantly

Bad symptoms:

- IO wait time increases, CPUs get blocked
- Node throughput goes down
- Hard to log into server and fix the problem! (Shell is unresponsive)

Seconds matter! Once thrashing starts, waiting 30 secs may make it irreparable.
Bad Swapping (aka “thrashing”) in Hadoop

Hadoop-specific symptoms of thrashing

- Tasks go slow on affected nodes → entire jobs slow down
- Can’t read data from affected DataNodes
- Easy to create many similar hungry tasks → problem can scale across the cluster very quickly
- Leads to “DataNode death” or entire HDFS unavailability
Wow, thrashing is bad… should I allow swapping at all?
Wow, thrashing is bad… should I allow swapping at all?

Yes!
Wow, thrashing is bad… should I allow swapping at all?

Without swap space, the same overload conditions that lead to swapping lead to the Linux Kernel OOM killer randomly killing processes.

- Good swapping is much preferable to this.

Little-known gotcha - don’t use swappiness=0 (anymore)

- Since kernel 2.6.32-303, swappiness=0 changed its meaning, and basically causes Linux Kernel OOM to be used instead of any swapping.
- Use swappiness>=1 instead. Cloudera recommends 10.
How to avoid thrashing?

Reduce number of containers on hosts
How to avoid thrashing?

Reduce number of containers on hosts

Lower throughput
A better, more dynamic approach

- Occasionally, over allocation of physical RAM can lead to swapping, then thrashing
- **Goal:** Keep high throughput without thrashing
- Obvious need: Run less containers when thrashing is likely/has happened
- Difficult need: Detect and act very rapidly (seconds)
Detect and Act: Overall Steps

- Observe metrics (per system, with fine granularity)
  - Pages swapped in and out
  - Free memory
- Detect thrashing (via supervised learning)
- Act automatically (without waiting for humans)
Thrashing Detection: Inputs and Output

Stream of system metrics → ThrashDetector → ThrashState (0, 1, 2)

<table>
<thead>
<tr>
<th>ThrashState</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No chance of thrashing</td>
</tr>
<tr>
<td>1</td>
<td>Higher chance of thrashing</td>
</tr>
<tr>
<td>2</td>
<td>Active thrashing</td>
</tr>
</tbody>
</table>
ThrashDetector: Overall Approach

Stream of system metrics

Estimate Memory Overuse → Memory Overuse

Estimate Potential Thrash State → Potential Thrash State

Remove False Positive → ThrashState (0, 1, 2)
Continuous pages coming in and out gives a good estimate of extra memory usage

- OverusedMemory = geo_mean(PagesSwappedInRollSum, PagesSwappedOutRollSum)
- Using geometric mean so the output is high when both inputs are high and the output units are still pages of memory.
ThrashDetector: Estimate Potential Thrash State

PotentialThrashState = min(floor(MemoryOverusePercent / MemoryOveruseThreshold), 2)
ThrashDetector: Removes False Positive

When enough memory is free, thrashing reduces without intervention.

**False Positive Removal:** Ignores potential thrashing if free memory is above a threshold.
Parameters to Learn

- **WindowSum**: Window to calculate rolling sum of pagesSwappedIn and pagesSwappedOut.

- **MemoryOverloadThreshold**: Memory overload threshold to generate potential thrashing state.

- **FreeMemoryThreshold**: Ignores thrashing if free memory is higher than this threshold.

- **WindowSmooth**: Window to smooth final thrashing states.
Hand labeling of thrashing events (supervised learning)

- 3.6M system metric tuples
  - Hand labeled every thrashing event in this time by expert system operators
- 5 second granularity data
  - Precision of label is critical
- Many factors considered by human experts in labeling:
  - Pages swapped in/out (up)
  - CPU usage (down)
  - Free memory incl buffers (down)
  - Metrics “gaps” (up)
  - IOPs on swap disk (up)
Learn Parameters: Minimize Score

- Score1 = Error in thrashing detection (did we detect the labeled thrashing event)

- Score2 = Time to detect thrashing (how quickly did we detect the event after it started)

Heuristic: Keep Score1 lower than a threshold, and then minimize Score2
Actions to Take: Scheduler Decision Process

ThrashState

No thrashing
- No action

Higher chance of thrashing
- Don’t add more containers on host

Active thrashing
- Don’t add more containers and kill running containers

Which container to kill:
- First, try to kill a mapper
- Then, try to kill a reducer
- Then, try to kill a Spark executor

Never kill an Application master

Minimum wasted effort
Simulation and Implementation

- Collected multi-customer production data.
- Learner implemented in Jupyter iPython notebook.
- Actual algorithm implemented in Java.
Results

- Detection algorithm running on all Pepperdata customers
  - Over 15k production nodes deployed

- Deployed clusters range from 10 to 1000’s of nodes

- No thrashing cases once action loop enabled.
Recommendations

- Use swap space
- Ensure swappiness > 0
- Be conservative in maximum RAM usage on nodes
- or-
- Use Pepperdata Cluster Optimizer
Thanks & Questions

Stop by Booth #1230

Also... Pepperdata is Hiring!

https://www.pepperdata.com/careers
https://www.linkedin.com/in/ssuchter/
https://www.linkedin.com/in/shkhrsgpt/