Far Memory-Aware Cluster Scheduler

Zhihong Luo

Introduction

▶ DRAM has become a bottleneck for scaling datacenters:
  ▶ End of Moore’s law —— stagnated cost reduction
  ▶ Big-data, analytics: explosive growth of memory demand

▶ Memory disaggregation allows compute nodes to access far memory at remote nodes
  ▶ Two main barriers to making far memory practical
    ▶ Fast swapping mechanisms needed to access far memory
    ▶ Far memory-aware cluster scheduler

▶ This project: design a far memory-aware scheduler that improves cluster efficiency.

Overview

▶ Data plane: far memory via swapping over RDMA enforced by Linux control groups
  ▶ Control plane: far memory-aware cluster scheduler
    ▶ Decide ratios to split between local memory and far memory
    ▶ Decide placement of jobs among servers

Fastswap RDMA swap system

▶ RDMA-based microsecond-scale swap system

Scheduler design

▶ Uniform policy
  ▶ Shrink all jobs uniformly up to a minimum ratio α

▶ Variable policy
  ▶ Allowing per-job minimum ratios
  ▶ Shrink proportionally according to minimum ratios

▶ Memory-time policy
  ▶ Asymptotic makespan of a workload:
    \[ \text{makespan} \approx \frac{\text{memorytime}}{\text{local mem} \cdot \text{utilization}} \]
    \[ \text{memorytime} = \sum_{i=1}^{N} \text{mem}_i \cdot r_i \cdot f(r_i) \]

▶ Main goal: save local memory-time product given limited far memory resource.

Evaluation

▶ Fastswap is implemented as a kernel module
  ▶ Control plane is comprised of a central scheduler and a per-node daemon.
    ▶ Scheduler uses gRPC to communicate with all daemons
    ▶ When the scheduler dispatches jobs to a daemon, the daemon creates a cgroup with a memory limit for it

▶ Evaluated on 1) a small testbed rack and 2) in simulation at full rack scale using a cluster simulator

▶ Testbed consists of 14 machines: 9 compute nodes, 1 scheduler, and up to 4 as memory servers

▶ Workloads generated from jobs in Table 1 with a random arrival order

![Figure 1: A 2GB job limited to 1GB.](image1)

![Figure 2: Performance degradation of our profiled applications. The constant local memory time line depicts A=B (Figure 3).](image2)

![Figure 3: How a job can reduce its local memory consumption by using far memory. A is the original memory-time product when no far memory is used. B+C is the new memory-time product, where B is the local portion, and C is the far portion of the product. r is the local memory ratio of the job.](image3)

![Figure 4: The percent improvement in workload makespan, relative to the Nofar configuration, for workloads with m2c ratio around 1.2](image4)

Table 1: Applications that comprise our workloads.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Memory (GB)</th>
<th># cpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>linpack</td>
<td>1.56</td>
<td>4</td>
</tr>
<tr>
<td>quicksort</td>
<td>6.05</td>
<td>1</td>
</tr>
<tr>
<td>kmeans</td>
<td>4.73</td>
<td>1</td>
</tr>
<tr>
<td>tensorflow-inception</td>
<td>2.07</td>
<td>2</td>
</tr>
<tr>
<td>memcached</td>
<td>12.00</td>
<td>2</td>
</tr>
<tr>
<td>spark</td>
<td>4.29</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Makespan improvement normalized to a Nofar (9 node cluster without far memory).

<table>
<thead>
<tr>
<th>m2c</th>
<th>Nofar</th>
<th>Far (+0%)</th>
<th>Far (+11%)</th>
<th>Far (+33%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.00</td>
<td>1.05</td>
<td>1.04</td>
<td>1.07</td>
</tr>
<tr>
<td>1.2</td>
<td>1.00</td>
<td>1.12</td>
<td>1.08</td>
<td>1.10</td>
</tr>
<tr>
<td>1.4</td>
<td>1.00</td>
<td>1.07</td>
<td>1.12</td>
<td>1.11</td>
</tr>
<tr>
<td>1.6</td>
<td>1.00</td>
<td>1.15</td>
<td>1.21</td>
<td>1.28</td>
</tr>
</tbody>
</table>