Bistro: Scheduling Data-Parallel Batch Jobs against Live Production Systems

http://bistro.io

Andrey Goder, Alexey Spiridonov, Yin Wang (Facebook)
Big Data and Hadoop
Facebook Data Store

MySQL

HBase

Haystack/F4
Facebook Data Store (cont’d)
Online vs. Offline

MySQL
PostgreSQL
Apache HBase
Hadoop

Haystack/F4
RocksDB
Bistro: Scheduling against a Variety of Online Systems
Outline

• Background
• The scheduling problem
• Bistro implementation
• Case studies
Data-Parallel Jobs

• Task-Parallel vs. Data-Parallel

Map-only jobs
The Scheduling Problem

Goal: maximize resource utilization (high throughput)
Challenge: minimize scheduling overhead at scale

/job_photo_process
1 lock, 100 IOPS, 1 Gbps

/job_volume_compact
1 lock, 200 IOPS, 0 Gbps
Existing Schedulers

• Online schedulers
  – E.g., load balancing, low latency, quality of service
  – Not designed for data-parallel batch jobs

• Offline schedulers
  – Can easily overload a data host
    • Focus on computation resources on worker hosts, ignoring resource constraints on data hosts
  – Little support for mutable data
  – Tightly integrated with a specific offline system

Fundamental problem: queue-based scheduling!
Head-of-Queue Blocking

Global queue?

One queue per shard?

T3 blocks the rest of the queue!

blocked by Host. Can run once T1 finishes.
How to Track Changes

Worker pool

FIFO task queue

<table>
<thead>
<tr>
<th>......</th>
<th>T4</th>
<th>T3</th>
<th>T2</th>
<th>T1</th>
</tr>
</thead>
</table>

New Vol
Vol C
Vol B
Vol A
Scheduling Algorithm – Brute-force

```python
while true:
    for j in jobs:
        for n in leaf nodes:
            if t has not finished and there are enough resources for j along the path to root:
                run(t)
```
Scheduling Algorithm – Subtree

when a task $t$ finishes at leaf node $n$, run:

$p = \text{the highest ancestor node of } n \text{ where } t \text{ consumes resources}$

run brute-force algorithm on the subtree at $p$

Enables parallel scheduling!
Micro-benchmark
Outline

• Background
• The scheduling problem
• Bistro implementation
• Case studies
Scheduling Modes

(a) multi/co-locate

(b) single/co-locate
Scheduling Modes (cont’d)

(d) single/remote

(e) multi/remote
Configuration

```
"nodes": {
  "levels": ["rack", "host", "volume"],
  "node_fetchers": [
    {"source": "haystack_racks"},
    {"source": "haystack_hosts"},
    {"source": "haystack_volumes",
     "prefs": {
      "volume_type": "PHOTO"
     }
    }
  ],
  "resources": {
    "rack": {"capacity": 20, "default": 1}
    "host": {"capacity": 200, "default": 100}
  }
},

"job->photo_process": {
  "cmd": "/photo_process",
  "args": {
    "model": "deep_face",
  }
  "enabled": true,
  "filters": {
    "host": {
      "blacklist_regex": "^haystack\([\d]+\)\.prn"
    }
  }
},

"job->volume_compact": {
  ...
}
```

Resource config

Job config
Scheduling Policies

• Round robin
• Randomized priority
• Ranked priority
• Long tail

while true:
    for j in jobs:
        for n in leaf nodes:
            if t has enough resources along the path to root:
                start(t)
Handle Model Updates

- Resource and job updates are common
- Periodically reload configuration, and update nodes
- Propagate changes to descendants
- Asynchronous model updates, scheduling, monitoring etc.
Worker Resource Management

run(t):
for w in workers:
if w has enough resources for t:
run(t, w)
return

• Just another layer of nodes in resource model
• Allows both resource constraints and placement constraints
• Can use randomized algorithms for scalability
Model Extensions

- Multiple resources per node
- Execute jobs on different levels
- Time-based jobs
  - Time nodes
- Different model partitioning schemes
- DAG model
  - Model replicas
“Reduce” Support

High performance intermediate data store, e.g., RocksDB
Adaptation to Live Traffic?

- Resource monitoring + Bistro model update
- What about spikey live workload?
  - Aggressive task preemption
  - Rapid cleanup
  - Making progress under frequent preemption?
- Conservative or simple scheduling policy wins
  - Overprovisioning
  - Day/night resource capacity adjustment
Model Limitations

• More complex resource constraints
  – Source destination
    • Break into two jobs
  – Network
Outline

• Background
• The scheduling problem
• Bistro implementation
• Use cases
Use Cases

<table>
<thead>
<tr>
<th>Online system</th>
<th>Example Job</th>
<th>Resource hierarchy</th>
<th>Replacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>DB Iterator</td>
<td>root-&gt;host-&gt;db</td>
<td>Proprietary scheduler</td>
</tr>
<tr>
<td>PostgresSQL</td>
<td>Migration</td>
<td>host-&gt;shard</td>
<td>Proprietary scheduler</td>
</tr>
<tr>
<td>BLOB storage</td>
<td>Photo/video processing</td>
<td>Host-&gt;volume-&gt;file</td>
<td></td>
</tr>
<tr>
<td>HBase</td>
<td>Data compression</td>
<td>host-&gt;region</td>
<td>Hadoop</td>
</tr>
</tbody>
</table>

- Our only general-purpose scheduler for non-HDFS systems
- Replacing Hadoop for many online HDFS systems
## Summary

<table>
<thead>
<tr>
<th>Application</th>
<th>Mode</th>
<th>Resource Model</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>level</td>
<td>resource type</td>
</tr>
<tr>
<td>UDB Migration</td>
<td>single/remote</td>
<td>1</td>
<td>root</td>
</tr>
<tr>
<td>UDB Scraping</td>
<td>single/remote</td>
<td>1</td>
<td>host</td>
</tr>
<tr>
<td>ODS Rollup</td>
<td>multi/co-locate</td>
<td>1</td>
<td>host</td>
</tr>
<tr>
<td>CheaFS Migration</td>
<td>single/co-locate</td>
<td>1</td>
<td>host</td>
</tr>
<tr>
<td>Photo Iterator</td>
<td>multi/local</td>
<td>1</td>
<td>host</td>
</tr>
<tr>
<td>Video Re-encoding</td>
<td>multi/remote</td>
<td>1</td>
<td>host</td>
</tr>
</tbody>
</table>
SQL Database

• Database Iterator
  – Motivating use case
  – Largest user base

• Database scraping
  – Shortest duration
Brute-force vs. Tree scheduling for scraping workload

(a) Makespan of scheduling one job

(b) Makespan of scheduling multiple jobs concurrently
Hbase: Problems with Hadoop

• Designed for offline batch jobs, no protection for live traffic

• Very little control over job execution
  – No canary, pause/resume, debug/re-execute checkpoint etc.

• Rigid computation model
  – All or nothing
  – Barrier of each phase
Hbase Compression (cont’d)

(a) HBase table A

(b) HBase table B
Conclusions

• Running jobs directly against production systems is the trend

• Tree-based scheduling allows
  – Hierarchical resource constraints
  – Efficient updates
  – Easy partitioning for flexible setup
  – Parallel scheduling

• Open source at http://Bistro.io