Container, System and Application Monitoring with Prometheus

Gareth Roy
HEPSYSMAN 16/1/2018
Large-scale cluster management at Google with Borg

Abhishek Verma†  Luis Pedrosa†  Madhukar Korupolu
David Oppenheimer  Eric Tune  John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

1. Introduction

The cluster management system we internally call Borg admits, schedules, starts, restarts, and monitors the full range of applications that Google runs. This paper explains how.

Borg provides three main benefits: it (1) hides the details of resource management and failure handling so its users can focus on application development instead; (2) operates with very high reliability and availability, and supports applications that do the same; and (3) lets us run workloads across tens of thousands of machines effectively. Borg is not the first system to address these issues, but it's one of the few operating with a set of qualitative observations we have made from operating Borg in production for more than a decade.

2. The user perspective

Borg's users are Google developers and system administrators (site reliability engineers or SREs) that run Google's applications and services. Users submit their work to Borg in the form of jobs, each of which consists of one or more tasks that all run the same program (binary). Each job runs in one Borg cell, a set of machines that are managed as a unit. The remainder of this section describes the main features exposed in the user view of Borg.
Google Borg Monitoring (Borgmon)

- Each service and task at Google exports metrics via HTTP.
- Easy to scrape via scripts or other automation tools.
- Google built a service called Sigma that would scrape, process and display detailed information on every job run.

Almost every task run under Borg contains a built-in HTTP server that publishes information about the health of the task and thousands of performance metrics (e.g., RPC latencies). Borg monitors the health-check URL and restarts tasks that do not respond promptly or return an HTTP error code. Other data is tracked by monitoring tools for dashboards and alerts on service level objective (SLO) violations.

A service called Sigma provides a web-based user interface (UI) through which a user can examine the state of all their jobs, a particular cell, or drill down to individual jobs and tasks to examine their resource behavior, detailed logs, execution history, and eventual fate. Our applications generate voluminous logs; these are automatically rotated to avoid running out of disk space, and preserved for a while after the task’s exit to assist with debugging. If a job is not running Borg provides a “why pending?” annotation, together with guidance on how to modify the job’s resource requests to better fit the cell. We publish guidelines for “conforming” resource shapes that are likely to schedule easily.

Borg records all job submissions and task events, as well as detailed per-task resource usage information in Infrastore, a scalable read-only data store with an interactive SQL-like interface via Dremel [61]. This data is used for usage-based charging, debugging job and system failures, and long-term capacity planning. It also provided the data for the Google cluster workload trace [80].
Prometheus is an open source version of the Borgmon idea.

Pull metric collection rather than push (push is available via a gateway).

Easy and simple to scale by adding Prometheus servers.

Uses a very simple exposition format.

Designed for “right now” monitoring, with a default retention of 15 days

Single go executable (or available via Docker)
Running with Docker

- Example running via Docker and Docker-compose
- Run a Prometheus endpoint as well as cAdvisor for monitoring local containers and Grafana for optional visualisation
- Persistence is via Docker volumes which extend beyond life of the container.
- Show how to monitor, containers, systems and legacy applications
cAdvisor

Isolation

CPU
Shares 1024 shares
Allowed Cores 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Memory
Reservation unlimited
Limit unlimited
Swap Limit unlimited

Usage

Overview

Processes

<table>
<thead>
<tr>
<th>User</th>
<th>PID</th>
<th>PPID</th>
<th>Start Time</th>
<th>CPU %</th>
<th>MEM %</th>
<th>RSS</th>
<th>Virtual Size</th>
<th>Status</th>
<th>Running Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>65534</td>
<td>3488</td>
<td>3470</td>
<td>11:38</td>
<td>1.20</td>
<td>0.10</td>
<td>110.20 MIB</td>
<td>147.57 MIB</td>
<td>Ss1</td>
<td>66:00:34</td>
<td>prometheus</td>
</tr>
</tbody>
</table>

Memory

Total Usage

Usage Breakdown

Network

Throughput

Errors
cAdvisor - Metrics (http://hostname:8080/metrics)

container_cpu_system_seconds_total
container_cpu_usage_seconds_total
container_cpu_user_seconds_total
container_fs_inodes_free
container_fs_inodes_total
container_fs_io_current
container_fs_io_time_seconds_total
container_fs_io_time_weighted_seconds_total
container_fs_limit_bytes
container_fs_read_seconds_total
container_fs_reads_merged_total
container_fs_reads_total
container_fs_sector_reads_total
container_fs_sector_writes_total
container_fs_usage_bytes
container_fs_write_seconds_total
container_fs_writes_merged_total
container_fs_writes_total
container_last_seen
container_memory_cache
container_memory_failcnt

container_memory_failures_total
container_memory_rss
container_memory_swap
container_memory_usage_bytes
container_memory_working_set_bytes
container_network_receive_bytes_total
container_network_receive_errors_total
container_network_receive_packets_dropped_total
container_network_receive_packets_total
container_network_transmit_bytes_total
container_network_transmit_errors_total
container_network_transmit_packets_dropped_total
container_network_transmit_packets_total
container_scrape_error
container_spec_cpu_period
container_spec_cpu_shares
container_spec_memory_limit_bytes
container_spec_memory_swap_limit_bytes
container_start_time_seconds
container_tasks_state
Prometheus Config

- Simply YAML file.
- This example we only specify scraping rules.
- Sources of machines to scrape can be:
  - azure
  - consul
  - dns
  - ec2
  - openstack
  - file
  - gce
  - kubernetes
  - marathon
  - nerve
  - serverset
  - triton
  - static

```yaml
0 global:
  1   scrape_interval: 15s
  2
  3 scrape_configs:
  4  - job_name: 'prometheus'
  5      scrape_interval: 5s
  6      static_configs:
  7          - targets: [ 'localhost:9090' ]
  8
  9  - job_name: 'cadvisor'
 10     scrape_interval: 5s
 11     static_configs:
 12        - targets: [ 'cadvisor:8080' ]
```
### Prometheus

#### Targets

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>State</th>
<th>Labels</th>
<th>Last Scrape</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://cadvisor:8080/metrics">http://cadvisor:8080/metrics</a></td>
<td>UP</td>
<td><code>Instance=&quot;cadvisor:8080&quot;</code></td>
<td>615ms ago</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>State</th>
<th>Labels</th>
<th>Last Scrape</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://kube003.beowulf.cluster:9100/metrics">http://kube003.beowulf.cluster:9100/metrics</a></td>
<td>UP</td>
<td><code>Instance=&quot;kube003.beowulf.cluster:9100&quot;</code></td>
<td>1.824s ago</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>State</th>
<th>Labels</th>
<th>Last Scrape</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://localhost:9090/metrics">http://localhost:9090/metrics</a></td>
<td>UP</td>
<td><code>Instance=&quot;localhost:9090&quot;</code></td>
<td>4.389s ago</td>
<td></td>
</tr>
</tbody>
</table>
Prometheus vs Grafana
node_exporter

- For system monitoring, Prometheus proved node_exporter
- Single golang binary that gathers a myriad of performance counters.
- Simple to run as a service (via systemd, init.d needs some work).
- Ansible script shown here.

```yaml
hosts: demo
tasks:
- name: Copy node_exporter binary to target systems
  copy:
    src: files/node_exporter
    dest: /usr/sbin/node_exporter
    owner: root
    group: root
    mode: 0755
- name: Copy node_exporter service file to target systems
  copy:
    src: files/node_exporter.service
    dest: /etc/systemd/system/node_exporter.service
    owner: root
    group: root
    mode: 0644
- name: Copy node_exporter environment file to target systems
  copy:
    src: files/node_exporter_env
    dest: /etc/sysconfig/node_exporter_env
    owner: root
    group: root
    mode: 0644
- name: Add node_exporter user
  user: name=node_exporter group=users shell=/sbin/nologin
createnewuser:
- name: Create textfile directory
  file:
    path: /var/lib/node_exporter/textfile_collector
    state: directory
    mode: 0755
- name: Enable and Start the node_exporter service
  service:
    name: node_exporter
    enabled: yes
    state: started
```
node_exporter

INFO[0000] Starting node_exporter (version=0.15.1, branch=HEAD, revision=ba5da2c9ae7f6209a88cb58676ba5ba029ad785)
INFO[0000] Build context (go=go1.9.2, user=root@b73d73f4fc5e, date=20171107-17:50:51) source="node_exporter.go:44"
INFO[0000] No directory specified, see --collector.textfile.directory  source="textfile.go:57"
INFO[0000] Enabled collectors: source="node_exporter.go:50"
  - cpu source="node_exporter.go:52"
  - filesystem source="node_exporter.go:52"
  - meminfo source="node_exporter.go:52"
  - conntrack source="node_exporter.go:52"
  - mdadm source="node_exporter.go:52"
  - hwmon source="node_exporter.go:52"
  - netdev source="node_exporter.go:52"
  - filefd source="node_exporter.go:52"
  - loadavg source="node_exporter.go:52"
  - ipv source="node_exporter.go:52"
  - edac source="node_exporter.go:52"
  - infiniband source="node_exporter.go:52"
  - xfs source="node_exporter.go:52"
  - bcache source="node_exporter.go:52"
  - textfile source="node_exporter.go:52"
  - uname source="node_exporter.go:52"
  - time source="node_exporter.go:52"
  - diskstats source="node_exporter.go:52"
  - entropy source="node_exporter.go:52"
  - wifi source="node_exporter.go:52"
  - timex source="node_exporter.go:52"
  - sockstat source="node_exporter.go:52"
  - stat source="node_exporter.go:52"
  - netstat source="node_exporter.go:52"
  - zfs source="node_exporter.go:52"
  - arp source="node_exporter.go:52"
  - vmstat source="node_exporter.go:52"
INFO[0000] Listening on :9100 source="node_exporter.go:76"
Prometheus Config

- We can add system monitoring by adding a simple rule to our prometheus config file.

- Adds another job, or we could run a second prometheus instance that only gathered system metrics.

- Easy to scale to any number of hosts by splitting up jobs across different Prometheus servers.

```yaml
0 global:
  1  scrape_interval: 15s
  2
  3 scrape_configs:
  4    - job_name: 'prometheus'
  5      scrape_interval: 5s
  6      static_configs:
  7        - targets: ['localhost:9090']
  8
  9    - job_name: 'cadvisor'
 10       scrape_interval: 5s
 11       static_configs:
 12         - targets: ['cadvisor:8080']
 13
 14    - job_name: 'host'
 15       scrape_interval: 5s
 16       static_configs:
 17         - targets: ['kube003.beowulf.cluster:9100']
```
Prometheus vs Grafana
## Application and Legacy Monitoring

```
svr009:~# condor_ls --vo

<table>
<thead>
<tr>
<th>Owner</th>
<th>Idle</th>
<th>Running</th>
<th>Removed</th>
<th>Completed</th>
<th>Held</th>
<th>Transfer</th>
<th>Suspended</th>
</tr>
</thead>
<tbody>
<tr>
<td>gridpp054</td>
<td>589</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ilc059</td>
<td>311</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>nst038</td>
<td>676</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ops061</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pheno016</td>
<td>5105</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pilatlas024</td>
<td>37</td>
<td>67</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pilcms008</td>
<td>152</td>
<td>75</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pilcms011</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pillhcb002</td>
<td>195</td>
<td>789</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pillhcb007</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>prdatlas006</td>
<td>567</td>
<td>280</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>prdatlas072</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>prdlhcb009</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sgmcms</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
textfile plugin

- It’s simple to add arbitrary metrics using the textile plugin to node_exporter.
- We can specify a directory and node_exporter will parse any files with a *.prom extension looking for metrics.
- Example parses our condor_ls tool and publishes running jobs.
- Exposition format can be found at: https://prometheus.io/docs/instrumenting/exposition_formats/
- Metric types:
  - Counter
  - Gauge
  - Histogram
  - Summary
  - Untyped
Prometheus vs Grafana
Monitoring VAC

- Wanted to monitor what VAC nodes were actually doing.
- `vacmon.gridpp.ac.uk`
- Fine for occupancy etc, but couldn’t see low level metrics.
- Run `node_exporter` on all machines, scrape via Prometheus.
- Initially attempt to replicate `vacmon` at local level.
Prometheus

- A good candidate for site monitoring.

- Things still to do:
  - CentOS6 init.d scripts

- Things I haven’t covered:
  - Recording
  - Alerting
  - Forwarding