Borg: A Cluster management system

Kubernetes' older sibling at Google

Aditya Shete · Follow
8 min read · Jun 13, 2022

Google is famous for its large-scale search engine and other highly available internet services such as Docs, Sheets, and Gmail, amongst many others. In addition to these internet-facing services, Google has internal tools and services that its employees use and develop, which must be hosted on servers.

We can model all possible services we would want to run under the abstraction of a task. So, with such vast workloads, it is unfeasible not to have an automated process to handle the provisioning and modification of computational tasks. In addition to task scheduling, we require the tasks to be highly available and resilient to failures. We look at Google's Borg, their internal cluster manager that affords them all these qualities.

Jobs and Tasks

Users submit their work to Borg in the form of *jobs*, each consisting of one or more tasks that run the same program (binary). A Borg job's properties include its name, owner, and the number of tasks. It can also have constraints on its resources which determine its required attributes, such as OS version and processor type. Tasks can also have attributes, such as resource requirements, hierarchy within the job etc.

A job can be assigned a priority order of production or non-production. A production job is generally a long-running process and is prioritized in the scheduling algorithm.

Cells

Each job runs in one Borg cell, a set of machines that are managed as a unit. Each cell belongs to a *cluster* of heterogenous machines, generally a data centre. A cluster contains a single cell plus other smaller cells reserved for testing or other purposes.

Borg cells run a heterogenous workload with two main parts. A long-running process/service which is meant for end-user-facing products. Batch tasks are relatively short-lived and less sensitive to performance fluctuations.

Borg masks this heterogeneity of machines by determining where a task can run in a particular cell, allocating it and its required resources. Further, after successfully installing the binary, Borg monitors the health of the task and attempts restarts in case of failures.

Naming

In addition to job scheduling, Borg assigns a Borg Name Service name for each job. This includes the job's cell name, job name provided by the user, task number, etc. The Borg algorithm maintains the task hostname and port in the local file system to look up the task for various RPCs. This could be routine health checks, routing load-balanced requests for the task or getting performance metrics.

Allocs

A Borg alloc is a reserved set of resources on a machine in which one or more tasks can be run. An alloc remains assigned regardless of whether tasks are run on it or not. Allocs can be used to set aside future requirements of resources, i.e. when a task is restarting. Allocs allow tasks to share the resources if assigned to the same alloc. If an alloc is reassigned to a different machine, all its tasks are rescheduled.

Preemption and Quota-checking

As the number of machines is finite, a state could arise where the number of tasks scheduled cannot fit on the compute resources available. Under such scenarios, Borg preempts tasks with lower priority in favour of scheduling tasks of higher priority. The preempted tasks are often rescheduled in the same cell and can potentially cause a cascade. Prod priority tasks cannot preempt other prod tasks.

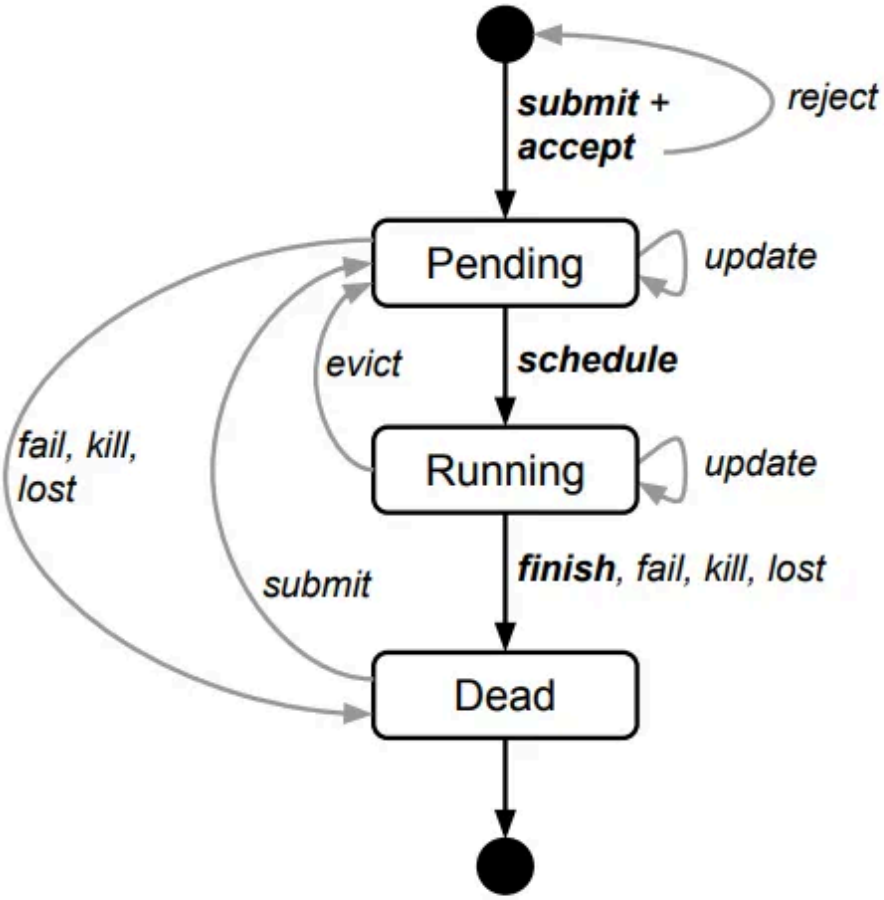
In addition to these scheduling level implementations, *quota-checking* at the time of admission is done to avoid scheduling tasks that cannot be provisioned. This quota checking is done over a resource vector for a task priority level, with each dimension being the max resource a task can ask for. Higher priority quotas are more expensive and are more in line with the actual resource limits. Lower quotes

can ask for increasing quotas, with the lowest priority having an unlimited quota. As task scheduling is achieved via priority, this quota allocation makes sense.

User — Flow

A user submits a job specified in a propriety declarative configuration language called BCL via remote procedure calls.

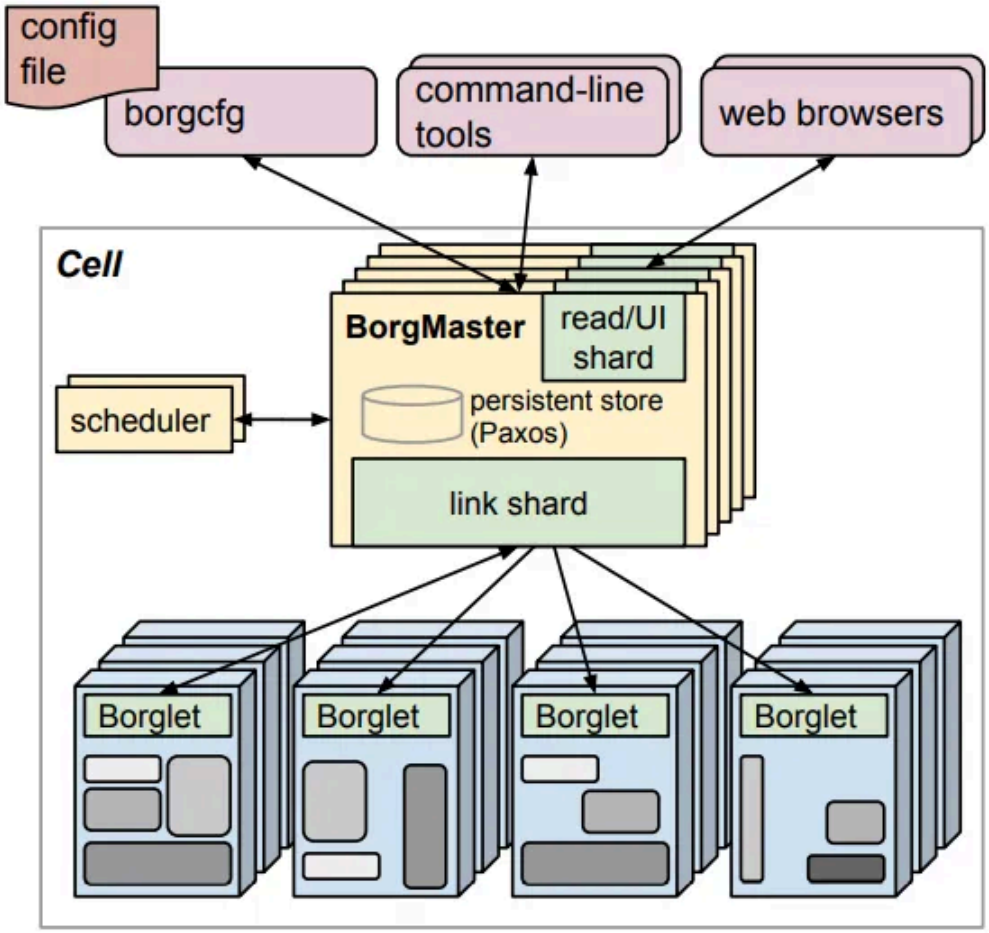
After a submission is accepted, the following state machine diagram shows the flow:



A user can modify all or some aspects of the job by simply pushing a new configuration to Borg and instructing it to update the job. Not all updates require the task to be killed; for example, resource reallocation could be achieved on the current machine. If such an update is not possible, the task is stopped and rescheduled.

Architecture

Each cell consists of a centralized control called Borgmaster and an agent process called Borglet that runs on each machine. Thus it is based on master-slave architecture.



A high level view of the Borg cluster manger

Borgmaster

The Borgmaster process comprises two sub-processes: the primary Borgmaster process and a separate scheduler. The Borgmaster process is tasked with the following objectives:

- Handle client requests such as creating a job, updating a job etc.
- Manage the state machine of each object in the cell; here, objects can be allocs tasks, machines and so on.
- Communicate with the Borglets for various health checks and performance metrics.

The Borgmaster is a replicated process, with a replication factor of 5, to achieve high availability and resilience. Each process contains a copy of the state of the cell, which is also stored in a Paxos-based store. An elected leader amongst these replicas is chosen at startup, which has the sole responsibility of state mutation such as job submission, termination etc. The Paxos algorithm also helps determine the leader after a master replica has failed.

When a job is submitted, the primary process records it persistently inside the Paxos store and adds the job to its pending queue. The scheduler scans this queue asynchronously to schedule jobs if the requisite resources are present. We look at the process in the next section.

The scheduler

The scheduler ranks jobs according to priority and schedules jobs according to this ranking, from higher to lower. For jobs in the same priority band, a round-robin strategy is utilized to avoid being stuck behind large jobs that fail to be allocated.

The scheduling algorithm has 2 parts: feasibility check and scoring. The feasibility check attempts to determine whether the job under consideration can be run in the cell, constrained by the job's requirements. In scoring, the algorithm determines whether the resources identified are a good fit for the job. This score of good-fit takes various things such as priority of the task and minimizing chances of preemption, locality of packages required, failure domains and balanced workloads across the machines into account.

Borglet

Each machine runs a Borglet process, which the Borgmaster uses to communicate jobs to run on the machines. Each Borglet process can do the following:

1. Start, stop task. Restart jobs in case of failures.
2. Manages local resources by manipulating OS kernel settings
3. Reports the state of the machine to the Borgmaster

The Borgmaster polls each Borglet periodically, for a few seconds, to update its local state of the cell and to send any outstanding pending requests. If a Borglet fails to respond to several such poll messages, effectively heartbeats, the machine is marked as down. Upon restoration of such Borglets, the Borgmaster sends a request to kill all processes, to stop duplicates.

Features

In this section, we look at some of the features that Borg supports.

Scalability

The Borgmaster process is shared across the replicas to achieve high performance and scalability. This means that each replica controls a shared partition of the cell.

Additionally, separate threads are created to communicate to Borglets and respond to RPCs.

As the most expensive parts of the scheduler are the feasibility and scoring, the scheduler is optimized in the following manner:

1. Score caching: Borg caches scores until there is a significant change in the properties of the machine or a task changes.
2. Equivalence classes: Tasks inside the job usually have identical requirements, which can be grouped under an equivalence class. Scores are only calculated once per equivalence class.
3. Relaxed randomization: Rather than calculate scores for all machines, Borg examines machines in random order until it has found "enough" machines to score, amongst which it chooses the best fit.

Availability

Borg handles the availability of tasks in the following manner:

- Automatically reschedule tasks, evicted or failed.
- Reduces failures by spreading tasks across failure domains.
- Limits task disruptions, avoid task: machine pairings causing failures.
- Recovers critical data from local disk by rerunning log saver task.

A task running on a machine will continue to run regardless of the Borglet process being up or failing. Hence, the task remains available. The master process is still required to add tasks to the cell.

Resource reclamation

One of the objectives of Borg is to maximize resource utilization which is the most efficient use of the resources available. All machines run a mixed workload with the prod and non-prod tasks simultaneously. These workloads work by evicting low priority or stalling the non-prod tasks in case of bursty loads of prod tasks. This use of resources which are potentially needed at full load but often remain unused during the life of the task is called resource reclamation. This is based on Borg's estimate for prod tasks that never run on reclaimed resources.

Isolation

Isolation is a security and optimization issue. Multiple tasks running on the same machine require process level and resource level isolation, so they do not interfere with one another.

1. The Linux chroot jail achieves security isolation for multiple tasks on the same machine.

2. Borg tasks are run inside a Linux cgroup-based resource container; the Borglet manipulates the resource container settings allowing fine control.

Concluding Remarks

The Borg task scheduler is an impressive piece of software that forms the base of the now-ubiquitous Kubernetes, with requirements to manage tasks of the order of thousands per second. The team's important observation is that *cluster management is more than task scheduling.* Other cluster service interactions, such as load balancing and naming, are also required to be managed.

In developing Kubernetes, specific abstractions such as jobs are replaced, instead opting for management of pods using labels. Additional abstractions such as service, which automatically load balances between pods, are introduced. Studying this system and comparing the Kubernetes concepts used today was interesting. I had to leave out certain topics, such as the scoring algorithm used, performance metrics and further discussions for the sake of brevity. For all those interested can find the reference[1].

References

[1] *Large-scale cluster management at Google with Borg*

K8s Kubernetes Google Algorithms

Follow

Written by Aditya Shete

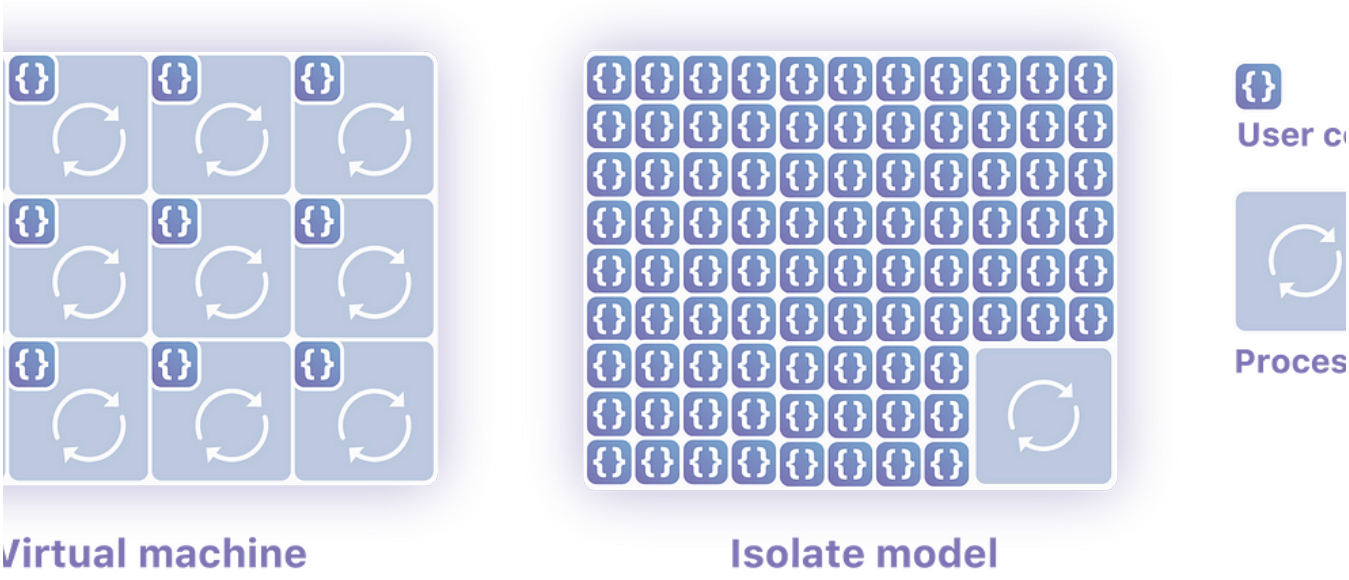
115 Followers · 33 Following

No responses yet

Yashavika Singh

What are your thoughts?

More from Aditya Shete



Aditya Shete

V8 isolates for Serverless Functions? A game changer

Recently I came across a very interesting article by Cloudflare, which described their use of V8 isolates in running serverless functions...

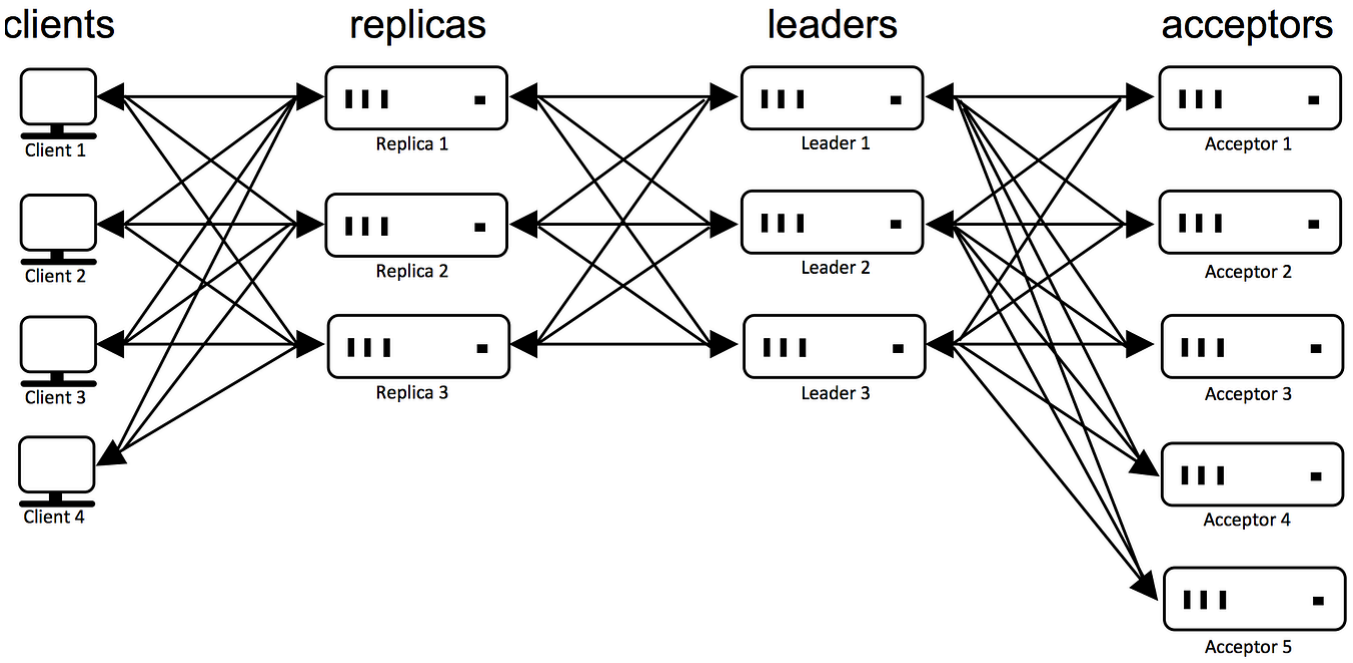
Oct 19, 2023 30

Aditya Shete

Consistent Hashing: Amazon DynamoDB (Part 1)

A look at how consistent hashing is used in DynamoDB

Feb 9, 2023 68 1



Aditya Shete

Notes on “Paxos made moderately complex”

Introduction

May 11, 2022 1

Aditya Shete

Vector Clocks: Amazon DynamoDB (Part 2)

A look at how vector clocks help reconciling divergent versions.

Feb 11, 2023 28 1

See all from Aditya Shete

Recommended from Medium

In Javarevisited by Rasathurai Karan

Java's Funeral Has Been Announced....☠️💻

Oh, Java is outdated! Java is too verbose! No one uses Java anymore!

Mar 6 1.5K 100

Kemila De Silva

Effortlessly Scale Kubernetes Workloads with Karpenter & KEDA! Cloud Scaling Series: EP03

This article is a follow-up edition in my series on Karpenter deployment. In this article, we will explore how KEDA helps us define scaling...

Mar 4 36

In Predict by Will Lockett

SpaceX Has Finally Figured Out Why Starship Exploded, And The Reason Is Utterly Embarrassing

This should never have happened.

Mar 1 10.4K 251

Ginger

I Pretended to Be a Man on a Dating Site—And I Hate What I Discovered

As a 23-year-old woman fascinated by human behavior (and, let's be honest, sometimes just bored and curious), I decided to conduct a...

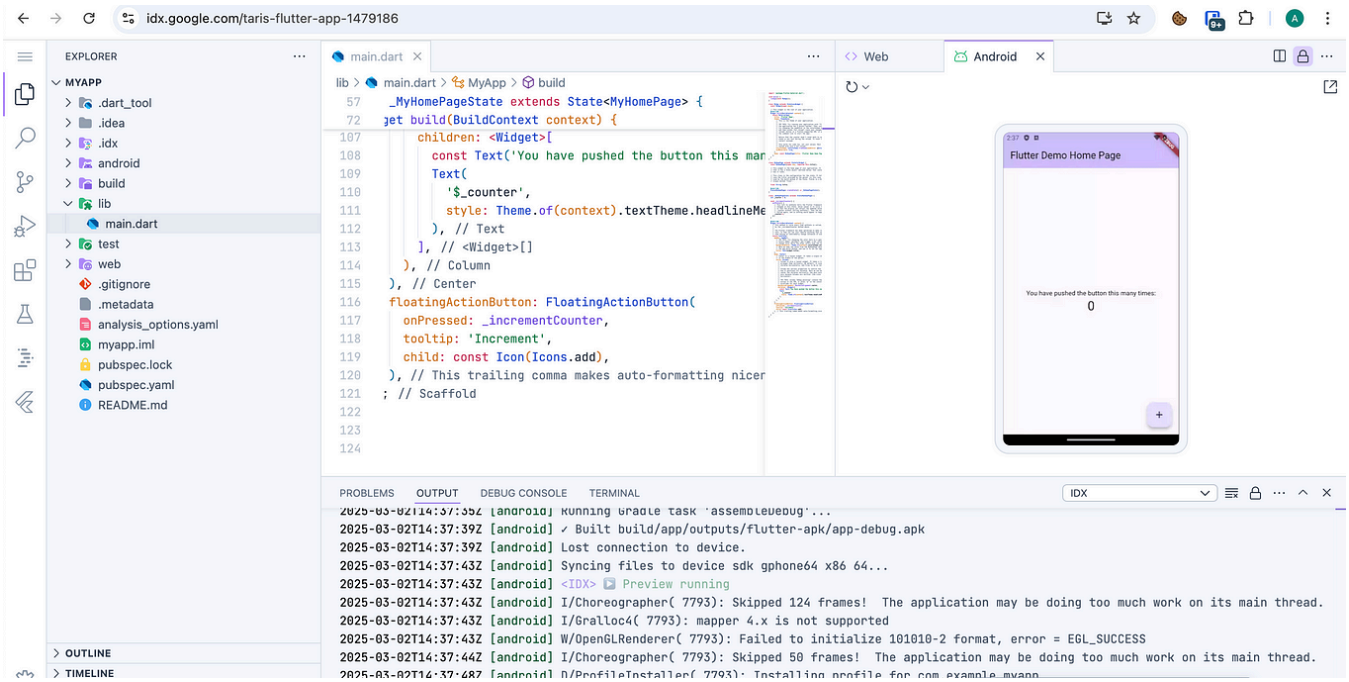
Mar 2 24K 693

Jessica Stillman

Jeff Bezos Says the 1-Hour Rule Makes Him Smarter. New Neuroscience Says He's Right

Jeff Bezos's morning routine has long included the one-hour rule. New neuroscience says yours probably should too.

Oct 30, 2024 25K 752



In Coding Beauty by Tari Ibaba

This new IDE from Google is an absolute game changer

This new IDE from Google is seriously revolutionary.

Mar 11 1.6K 102

See more recommendations