Student Name: Nikhil Doifode
Student ID#: 112714121

Review of “Large-scale cluster management at Google with Borg”

Borg is developed at Google for internal use on large, shared clusters. It automates application deployment and resource management on large clusters. It supports almost all workloads at Google, like Gmail, MapReduce, etc. It has inspired several similar open-source systems like Apache Mesos, Google Kubernetes, etc.
Google has several different types of workloads like from Gmail which should have very low latency and MapReduce which should have very high throughput. Also resource sharing needs to be optimal to save money since google needs many servers to serve millions of users concurrently. Also the cluster manager must have provision for peak load since, most of the time, load very less than peak hence, machines are poorly utilized.

Borg was designed in such way that it provides following benefits:

1. Hide details of resource allocation and failure handling
2. Very high reliability and availability
3. Efficiently execute workloads over 10,000+ machines

Following terminologies are used in Borg:

1. Borg supports all workloads like:
    - Storage servers (GFS, BigTable, MegaStore, Spanner)
    - Production front-ends (Gmail, Docs, Web Search)
    - Batch analytics and processing (Maps tiles, Crawling, Index Update, MapReduce)
    - Machine learning (Brain / TensorFlow)

2. Clusters, cells, job, tasks and containers:
    - Work is submitted in form of Job
    - Each job is number of tasks all executing the same program (binary).
    - Machine sharing is done at process level by performance isolation called containers.
    - Jobs are executed in cells.
    - Each cell is set of machines managed as a whole.
    - Interaction between users and jobs is done mostly via command line (RPC)

3. Resources Allocation (Allocs):
    - Resource allocation is done at machine level for CPU, RAM, I/O, etc.
    - Alloc set is like job, it is groups of allocs that reserve resources on a set of machine

4. Priority, quota and admission control:
    - Usage of quotas and priorities if more work than admissible shows up
    - Quota is used to decide which jobs to admit for scheduling.

- Quota is expressed as a vector of resource quantities (CPU, RAM, disk, etc.) at a given priority, for a period of time (typically months).
- Every job has a priority, a small positive integer.
- A high priority task can obtain resources at the expense of a lower priority one, even if that involves preempting (killing) the latter.

5. Naming and monitoring:
    - BNS (DNS) is used for Borg jobs for each task that includes the cell name, job name, and task number.
    - Borg writes the task's hostname and port into a consistent, highly-available file in Chubby with this name, which is used by our RPC system to find the task endpoint.
    - Ex: the 1st task of job foo owned by user joe will be reachable via **1.foo.joe.cc.borg.google.com** in cell **cc**
    - Borg also writes job size and task health information into Chubby whenever it changes, so load balancers can see where to route requests to.
    - Borg monitors the health-check URL and restarts tasks that do not respond promptly or return an HTTP error code.

Strengths:

1. Jobs with constraints and preference (dependency, environment, etc.) and other task properties are configured and handed to suitable clusters for better locale or performance.
2. Built-in HTTP server for almost every Borg tasks to publish information about the health of the task and performance metrics.
3. Sigma service provides web-based UI for processing logs and debug information.
4. The task/job control is easy and straightforward, which is quite similar to Linux processes or SLURM resource management.
5. Scheduling optimizations and relaxed randomization.
6. Equivalence classes become useful when assigning jobs with identical constraints.

Weakness:

1. Borg chooses containers over Virtual Machines for the sake of performance, but it might suffer from security and container management issues.
2. There is no performance comparison between Mesos or Kubernetes.
3. No discussion about the structural difference between the similar resource management systems.

Questions:

1. How is Borg integrated in a cluster?

Ans: Borg is integrated in cluster with help of BorgMaster and Borglet. A cluster can hosts one or multiple Borg cell. Each Borg Cell consists of several thousands of machines. Each machine in the cell runs the Borg agent called Borglet.

The BorgMaster coordinates all activity in the cell. It is implemented as a five-machine Paxos replicated group. BorgMaster is central to Borg as it holds cluster state. The BorgMaster coordinates with a scheduler to schedule tasks..

Following figure depicts this:

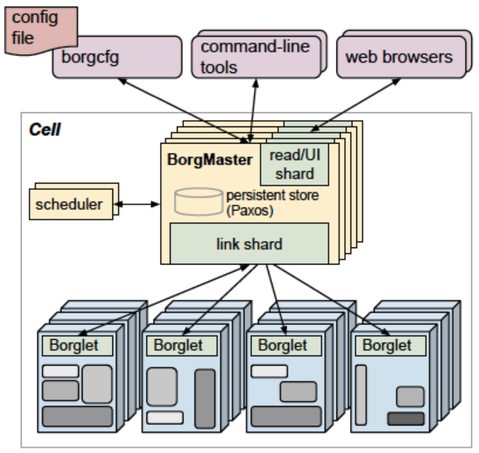


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

At Borg work is submitted in form of Job where each job is number of tasks all executing the same program (binary). Resource sharing is done at process level by performance isolation by use of allocs called containers. Alloc set is like job, it is groups of allocs that reserve resources on a set of machine.
Borg also uses quotas and priorities if more work than admissible shows up. Quota is used to decide which jobs to admit for scheduling. Every job has a priority, a small positive integer. A high priority task can obtain resources at the expense of a lower priority one, even if that involves preempting (killing) the latter.
BNS is used for Borg jobs for each task that includes the cell name, job name, and task number. Borg writes the task's hostname and port into a consistent, highly-available file in Chubby with this name, which is used by our RPC system to find the task endpoint. E.g.: the 1st task of job foo owned by user joe will be reachable via 1.foo.joe.cc.borg.google.com in cell cc

2. How do Borglet and BorgMaster work in Borg?

Ans: BorgMaster does following tasks in Borg. It handles RPC from clients, create machine, store states and communicate with Borglets. Logically BorgMaster is a single process but actually has 5 replicas. Each replica holds memory copy of states of cell by use of Paxos to maintain consistent states and elect master. All submitted jobs to Borg are recorded in Paxos store. They are scanned asynchronously by Scheduler. BorgMaster features one scheduler and it also considers priority levels and round robin inside a level.
There are two phases to scheduler:

- Feasilbility checking: Find candidate machines for tasks

- Scoring: Pick machines in feasibility test. Tries to pick machine with copies of task's packages already present and minimizes number of preemptions if cell is loaded and task of high priority

Borglet does following tasks in Borg. It is a local agent of Borg running on every machine
It start and stop tasks, manages local resources and manipulates local OS settings.
BorgMaster pulls data from Borglet every few seconds. Pull tasks are shared between BorgMaster replicas that reports to master only if it is different from previous report.
If Borglet is unable to communicate it continues work. BorgMaster reschedules tasks on different machine and kills them if Borglet re-appears.

Each job submitted to Borg goes through following phases:

i. User submits a job, which consists of one or more tasks
ii. Borg decides where to place the tasks
iii. Tasks start running, with Borg monitoring their health
iv. Runtime events are handled
    a. Task may go "pending" again (due to machine failure, preemption)
    b. Borg collects monitoring information
v. Tasks complete, or job is removed by user request
vi. Borg cleans up state, saves logs, etc.

Following figure depicts this:

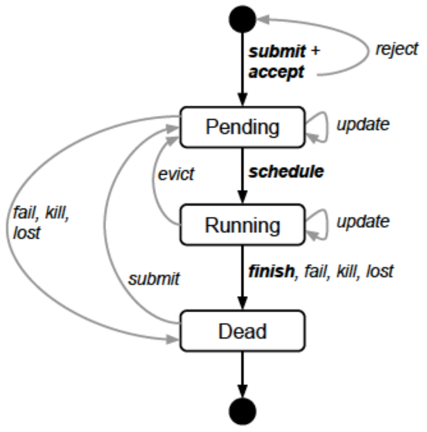


Figure 2: The state diagram for both jobs and tasks. *Users can trigger* ***submit, kill,*** *and* ***update*** *transitions.*

3. What metric is been used to evaluate the policy choices?

Ans: To measure utilization i.e. to check whether Borg's policy is the best for utilizing clusters several approaches were tried. E.g.
Average utilization - But it doesn't take care of load spikes and batch jobs.

So cell compaction was used.

Cell compaction:

1. Takes a workload (production or non-production)
2. Take the smallest cell, try to fit workload with the scheduling algorithm.
3. Remove machines randomly from a cell to maintain cell heterogeneity.

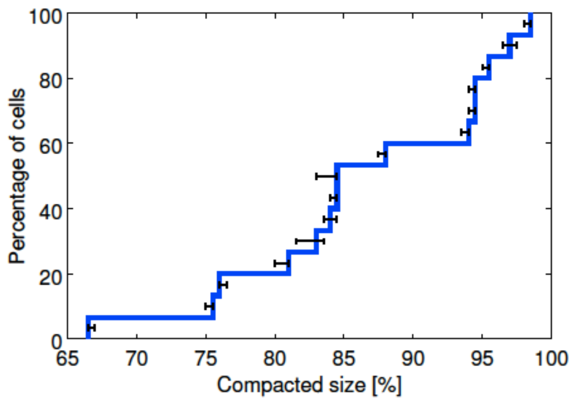
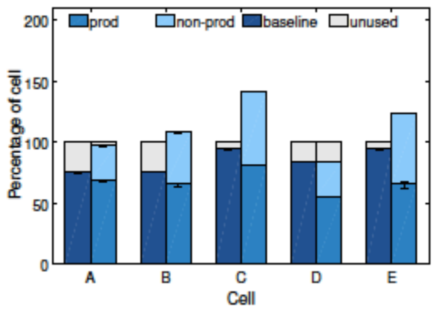
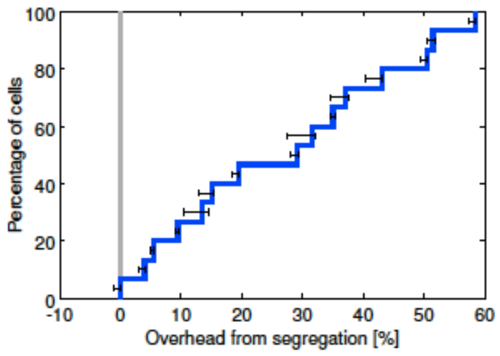


Figure 4: The effects of compaction. *A CDF of the percentage of original cell size achieved after compaction, across 15 cells.*

However, segregating production and non-production would require 20-30% more machines because latency sensitive (production) workload reserves additional resources for spikes



(a) *The left column for each cell shows the original size and the combined workload; the right one shows the segregated case.*



(b) *CDF of additional machines that would be needed if we segregated the workload of 15 representative cells.*

References:

Large-scale cluster management at Google with Borg – Abhishek Verma,…