Md Majid Jahangir | SBU ID - 112077145
AMS 560 - Big Data Systems, Algorithms and Networks

Assignment 5 (Paper Reading)
Large-scale cluster management at Google with Borg

Motivation : Since last decade data-centers have become prominent across the world, where each datacenter spans acres of lands and houses hundreds of clusters with each cluster having tens of thousands of machines. Such a data-center environment runs hundreds of thousands of jobs from several hundreds of applications all together. Managing such hundreds of thousands of jobs at this scale in terms of admission control, efficient task packing, machine sharing with process level performance isolation in order to achieve high utilization had become a daunting task. Also, demands for running applications that would require high availability having runtime features that would minimize fault-recovery time and have scheduling policies that would reduce the probability of failures had increased in such a large scale environment. Such a situation demanded a large scale cluster manager that would run and manage such hundreds and thousands of jobs and would offer a specification language, name service integration, real time job monitoring, etc. This paper [1] introduced such a cluster manager called Borg which has been the powerhouse of data-centers at Google for years.

Prior Solutions: Some of the prior solutions solving the same problem as that of Borg are Apache Mesos, YARN, Facebook's Tupperware, Twitter Aurora, etc. Apache Mesos usually splits the resource management and placement functions between a central resource manager and frameworks such as Hadoop and Spark by using an offer based mechanism. Its functionality is mostly similar to Borgmaster (covered below) minus the latter's scheduler. On the contrary, Borg centralizes such placement functions by using a request based mechanism. YARN, another cluster manager which is centred mostly towards Hadoop provides each application with a manager that demands for the resources required by the application with a central resource manager. This model is almost as same as the model used by Google MapReduce jobs to demands resources from Borg and only recently YARN has become fault tolerant and is proposed to support multiple resource types, priorities and advanced admission control. Facebook's Tupperware is an another system similar as Borg which is used for scheduling cgroup containers on clusters. Twitter's Aurora is another scheduler as same as Borg used for running long running services on top of Mesos providing a configuration language and state machine similar to Borg. Quincy, another similar solution uses a network flow model in order to provide data-locality aware scheduling and fairness for DAGs on clusters, however Borg uses quota and priorities in order to share resources between users. Quincy scales to hundreds of nodes in a cluster whereas Borg scales to tens of thousands of nodes in a cluster; Quincy handles execution graphs directly however Borg does not do that and this functionality is built separately on top of Borg.

Key Ideas of Proposed Solution: Borg is the Google's large scale cluster manager. The key ideas of the proposed solution are -

1. **Jobs and tasks** - Jobs, which are a collection of tasks, are submitted to Borg to be run in a single cell, most of which lives inside of a single cluster. The jobs submitted to Borg are either high priority latency sensitive jobs or low priority batch jobs. Each of the jobs in Borg have properties such as name, owner and the number of tasks it has. They can also have constraints which expresses its tasks to run on specific machines with specific architectures. Each of the task is mapped to a set of Linux processes running in a container on a machine and tasks have properties such as resource requirements, etc. These tasks or programs are statically linked because of which all the dependencies on runtime environment is reduced.
2. **Naming and monitoring** - Borgs also provides a naming and load balancing service and a web UI service called Sigma to examine state of jobs and tasks. More details covered in "strengths" section points numbered 4 and 5.

3. **Alloc; Priority and quota** - All the jobs are assigned a priority and they operate within quota limits and all the resources are bundled into allocs (alloc is a reserved set of resources on a machine in which one or more tasks can be run) in which multiple tasks can run.

In order to provide high availability, Borgmaster is Paxos replicated over 5 machines. Replicas will only serve read only RPC calls in order to reduce the workload of Borgmaster leader. Also, periodic snapshots or checkpoints are taken to ensure fault tolerance. The scheduler is decoupled from Borgmaster and is loosely synchronised with the state of Borgmaster. The scheduler operates on a cached copy of cell state and makes decisions based on that. If the decision made by scheduler is found to be not feasible by Borgmaster, it will reschedule the job.
A hybrid packing scheduling model is used instead of tight packing scheduling because tight packing seems to be too strict to accommodate high loads from Borg clients. For this purpose, Borg uses priorities for tasks because of which whenever a node runs out of resources for its assigned tasks, tasks which are prioritized lower in that machine are killed and are placed in the scheduler's pending queue.
Borg also provides a declarative job specification language in order to help users manage their jobs. Several other detailed and key concepts of the solution are discussed in key questions and strengths sections below.

Performance: To evaluate the scalability of the system, the authors added separate threads to talk to Borglets and to respond to read only RPCs and sharded the placement functions across the 5 Borgmaster replicas because of which 99 percentile of response time of UI is below 1 second and 95 percentile of the Borglet polling interval is below 10 seconds. To test the availability, Borgmaster uses a combination of techniques that enables it to achieve 99.99% availability.

How is Borg integrated in a cluster? [Key Question]: A cluster usually hosts one large Borg cell, where a cell is nothing but a set of machines, a centralized controller known as the Borgmaster and an agent process known as Borglet that runs on each machine in a cell. Each of the cell's Borgmaster comprises two main processes, first, the main Borgmaster process and another scheduler process. The Borgmaster is logically a single process which is Paxos replicated 5 times. Each replica of Borgmaster maintains a copy of the state of the cell in memory and the state is also maintained in a highly available distributed Paxos based data store on replica's local disks. Out of the 5 replicas, a master is elected using Paxos whenever a Borg cell is brought up or whenever an elected master has failed. The elected master serves as Paxos leader and state mutator and manages all operations such as communication with Borglets, submission of jobs or termination of jobs or tasks on a machine. The Borgmaster's state at any instant of time is called a checkpoint which is snapshotted in periodic intervals which helps in restoring the state of Borgmaster to a past state for debugging or during failures, when replicas recovers in order to resynchronise its checkpointed state from other replicas. A Borgmaster simulator called Fauxmaster is also used which contains the complete copy of the Borgmaster production code. Fauxmaster accepts RPCs in order to make state machine changes and perform other tasks such as scheduling pending tasks. Such a setup is also used during debugging by interacting with Fauxmaster as if it was a real Borgmaster with some simulated Borglets that replays real interactions using checkpointed files.
Whenever a job is submitted, the Borgmaster will record the job persistently in the Paxos data store and will add the tasks of the job to the pending queue. This pending queue is scanned and searched asynchronously by the scheduler in order to assign the tasks to machines which has sufficient resources that are available to meet the tasks' requirements. The tasks has priorities and they are scanned in the queue from high to low priority in a round robin manner to ensure fairness and to avoid starvation of small jobs. The scheduling algorithm does two key operations - feasibility check to find a set of machines that meets the requirements of tasks in terms of available resources; and, scoring to determine the score of each feasible machine to pick the best feasible machine out of the entire feasible set. The scoring criteria takes into account user defined preferences and several other built in criterias such as minimizing the number of

preempted tasks, picking machines which already has a copy of task's packages, etc. Some of the other details are covered in "How do Borglet and Borgmaster work in cluster" section.

What metric has been used to evaluate policy choices? [Key Question]: Cell compaction has been used as a metric to evaluate policy choices. For any given workload, the metric tries to find how small a cell could be in order to run that workload. It would find how small a cell could be to fit a workload into it by removing machines from cells until the workload no longer fitted the cell, and it would do so by repeatedly repacking the workload from scratch in order to ensure that not an unlucky configuration was chosen for the workload.

How do Borglet and Borgmaster work in a cluster? [Key Question]: From the previous section about integration of Borg in cluster, we know what a Borgmaster is. A Borglet is a local Borg agent that runs on every machine locally in a cell. Its role is to start and stop the tasks in each machine and to restart them in case of failures; manage local resources by manipulating OS kernel settings; reporting the state of the machine to the Borgmaster and other monitoring tools. The Borgmaster usually polls each Borglet periodically in few seconds in order to get the machine's current state and to send any outstanding requests to the machine. The elected master Borgmaster is also responsible for preparing messages in order to send them to Borglets and to update the cell's state with the Borglet's responses. Also for scalability, each Borgmaster replica runs a stateless link shard to handle communication with some of the Borglets. The Borglet will always report its full state to Borgmaster replicas via this shard link however the shard link will compress the information by reporting only the differences to the state machine which helps in reducing the update load at the elected master.
Whenever a Borgmaster does not receive responses to several poll messages to a Borglet, its machine is marked as down and any tasks that were running on that machine are rescheduled to be run on other machines and when the communication is restored between them, the Borgmaster tells the restored Borglet to kill all the tasks that have been rescheduled in order to avoid duplicates. Also, even if a Borglet loses contact with the Borgmaster, it continues to work normally such that all the running tasks are up and running even if all the Borgmaster replicas fail.

Strength :

1. One of the key strength of Borg is its scheduling optimizations. It uses score caching in order to store the evaluation of the performance of a machine. The performance of such a machine will be updated only when a new task or job is executed. Also, by using equivalence classes, it assigns jobs having identical constraints efficiently. And finally by using relaxed randomization, the scheduler, instead of trying to find the best machine, will find a good enough machine randomly in order to assign it the task.
2. Borg is highly scalable where a single Borgmaster can manage cells having hundreds of thousands of machines and nodes; and several of the cells have a task arrival rate of greater than 10000 tasks per minute and by using threading and sharding of placement functions between Borgmaster replica, Borg guarantees 99 percentile of response time of the UI to be lesser than 1 second and 95 percentile of Borglet polling interval to be lesser than 10 seconds.
3. The Alloc abstraction used by Borg is another very beneficial design feature which has led to logsaver pattern and another design pattern where a data-loader task periodically updates the data used by a web server. Such an abstraction provided by Allocs allows to develop helper services parallely by different teams.
4. Every task running under Borg contains a built in HTTP server which is used to publish health information about the tasks and other thousands of performance related metrics such as RPC latencies. It becomes easier to monitor such tasks by pinging their health check URL and tasks are restarted that do not respond promptly thus helps a lot in debugging
5. Also, a service called Sigma is provided which gives a web based UI to examine the state of the jobs running under Borg. It also examines each cells and can also be used to examine the resource usage of individual jobs and tasks. Also, large amount of logs

are generated by applications which are rotated to avoid running out of disks and are preserved for some time even after the termination of a task which helps in debugging. Borg also records all the job submissions and task events and per task resource usage in a read only data store called Infrastore which provides an interactive SQL based interface via Dremel. The data stored here can again be used for debugging and tracking system failures. All of these features of Borg provides a great utility in debugging the behavior of Borg and the jobs and tasks submitted to it.

6. The design implementation of Borg has been such that any service could be added on top of it easily adding more features to the whole ecosystem. Borg is designed as more of a kernel at the heart of a distributed system because of which over time, its scheduler and Sigma (the web based UI) has been splitted off into separate processes and has also enabled to add new features such as admission control, vertical and horizontal autoscaling, cron job submissions, workflow management, etc. Therefore such a design has enabled Borg to enhance its feature set and scale the workload without compromising on performance.

Weakness :

1. In Borg, jobs are restrictive as the only available grouping mechanism for any tasks. It has no way to manage a multi-job service as a single entity and also has no way to refer to related instances of a service. It is also not possible to refer to an arbitrary subset of jobs which can lead to problems such as inflexible semantics in order to roll updates and to resize the job. Therefore Borg becomes quite inflexible when it comes to any group related operations as there is not way to describe any relation between two tasks.
2. Usage of one IP address per machine also is a huge weak point for Borg. All tasks on a machine in Borg uses the same single IP address of their host and shares the host's port space. This leads to problems such as the task must declare the number of ports needed by them beforehand and the tasks are often instructed which port to use when they start; also, Borg has to schedule ports as a resource and Borglet has to enforce port isolation.
3. Borg has been designed to optimise performance for power users and not for casual users. And therefore in order to make it useful for casual users, several automation tools and services needs to be built in order to run it separately on top of Borg.

Follow Up Works :

1. ConHub: A Metadata Management System for Docker Containers [5] - Since many years, organizations have been using virtualization to simulate their workloads and it would mean running those applications in a virtual machine i.e. a hardware virtualization. However, most recently virtual machines are being replaced rapidly by containers i.e. an operating system virtualization as seen by rapid emergence of Docker. It is observed that a containerized environment generates a large amount of metadata which can facilitate proper management of containers. This paper introduces ConHub, a PostgreSQL based container metadata management system which supports a language CQL that supports docker commands; has a user friendly interface for querying and visualizing container relationships; and how CQL can be used to formulate queries to facilitate container management.
2. Learning to diagnose stragglers in distributed computing [6] - The paper talks about how a large job is divided into several small tasks for parallel execution in a distributed environment; but due to some reasons there are some tasks known as stragglers which are slower than others and delays the completion of such jobs. The paper proposes a new machine learning approach to identify and diagnose the stragglers. The paper talks about using an unsupervised clustering method to group the tasks based on their execution time in order to identify stragglers and then use a supervised learning algorithm to learn diagnosis rules inferring the stragglers with their resource assignment

and performance data. The paper used Google's Borg's traces to perform experiments to demonstrate that their method generates easy to read rules with decent performance in predicting stragglers.

References :

1. Large-scale cluster management at Google with Borg - Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, John Wilkes, Google Inc
2. https://mwhittaker.github.io/papers/html/verma2015large.html
3. https://xduan7.com/2016/04/12/paper-review-large-scale-cluster-management-at-google-with-brog/
4. http://muratbuffalo.blogspot.com/2017/02/large-scale-cluster-management-at.html
5. ConHub: A Metadata Management System for Docker Containers: Chris Xing Tian, Aditya Pan, Y.C. Tay - https://dl.acm.org/citation.cfm?id=2983331
6. Learning to diagnose stragglers in distributed computing: Cong Li, Huanxing Shen, Tai Huang, Intel Corporation - https://dl.acm.org/citation.cfm?id=3019079