A Survey On Efficient Container Orchestration Tools And Techniques In Cloud Environment

Hritwik Bairagi, Uday Chourasiya, Sanjay Silakari, Priyanka Dixit, Smita Sharma

Abstract: Containers are used to deploy large and diverse workloads resulting from applications over virtual machines which can be both, local as well as geographically distributed. Containers have less starting time than VMs and are manufactured to work with lower memory resources. The microservice architecture allows applications to be broken into distinct modular services which are packaged into separate containers in a way that they require to access certain binaries and library files specific to the application. There is a mass migration of application developers towards container platforms which has contributed to need of container orchestration softwares. This paper discusses the conceivable solutions to the shortcomings in present container orchestration methodologies. It discusses various implementations carried out in the context of container orchestration.

Keywords: Cloud Computing, containers, orchestration, virtual machine, microservices, Docker, Kubernetes

1 INTRODUCTION

With regards to its numerous advantages over the VM centric architectures, containers have been now become the default deployment platform for large scale applications and majority of the businesses are migrating towards container technology. With growing interest of the industry towards microservices and containers go hand – in – hand, a need for developing sophisticated management systems arises. In the cloud computing scenario, Container as a service (CaaS) is emerging as leading type of service among other traditional services. Containers facilitate the packaging, deploying and management of large, complex applications. The containerization of applications is one of the technologies enabling microservices architecture [2]. They package a software into small manageable sub-applications which are self-contained and support deployment on different platforms. The container technology has gained tremendous recognition amid major and young enterprises which deploy their web applications on cloud. Companies which previously relied upon VM based deployments are now shifting to container architecture. Containers promote a few features of the operating system kernel i.e. namespaces and control groups (cgroups). These are explicitly Linux kernel features which enable isolation of processes and when these kernel features were addressed to separately in order to develop lightweight, OS level virtualisation, Docker was developed. Namespaces are responsible for separation of processes for example, the process id (pid) namespace.Control groups or cgroups are employed to limit and control resources.

- *Hritwik Bairagi,1Department of Computer Science and engineering, UIT-RGPV, Bhopal, M.P., India. hbhritik@gmail.com*
- *Uday Chourasiya2,Department of Computer Science and engineering, UIT-RGPV,Bhopal, M.P., India. uday_chourasia@rediffmail.com*
- *Sanjay Silakari,Department of Computer Science and engineering, UIT-RGPV,Bhopal, M.P., India. ssilakari@yahoo.com*
- *Priyanka Dixit4Department of Computer Science and engineering, UIT-RGPV,Bhopal, M.P., India. priyanka.dixit@yahoo.com*
- *Smita Sharma55Department of Computer Science and engineering, UIT-RGPV,Bhopal, India sharmasmita34@gmail.com*

Another component used is SELinux – Security enhanced Linux, which is employed to insure separation amongst the

host and the container and between respective containers. With The release of Docker in March 2013, it soon became a subject of mass adoption and a buzzword for containerization in the industry (Rodriguez et al, 2018). Docker was developed on LXC but later Docker Inc. created its own unique runtime called 'libcontainer' in order support Go language bindings [6]. Docker is designed in way that makes container technology fast, easy to use and promote service oriented architecture (SOA) and especially the microservice architecture style. Docker [1] is the most widely used and the industry standard for containerization. Docker containers consist of layered images and binaries packed together into a single package. The base image contains the operating system of the container. Docker environment has four integral parts namely the docker client, docker host, docker engine, and the dockerfile. The Docker client is the UI for docker. It handles the dockerfile which is used to provide instructions to build a docker image and enables the user to communicate with docker. The docker client interacts with docker daemon which executes commands sent to the docker client. Containers are classified on what they contain. An application container runs a complete application or a component of an application. As the containers share the same host operating system kernel, multiple application components can run on the same virtualized machine. A container also solves the dependency hell problem. Another type of containers are the System containers which pack the whole operating system. Unlike Virtual Machine images containers are lightweight and can support different platforms. Some of the most popular platforms being Docker, Amazon Elastic Container Service (Amazon ECS), OpenShift, Google Kubernetes Engine, IBM Cloud Kubernetes Service, Azure Container instances etc. An important problem to address in this context is the scheduling or placement of containerized applications on the available hosts. As applications are ready for deployment, the orchestration system must allocate them with least delay on an available resource taking into account the application's constraints and maximizing the utilization of the resources to reduce to the operational cost. This should also be done while considering factors such as the capacity of the available machines, application performance and Quality of Service (QoS) [5] requirements and fault-tolerance. This paper presents a study on

container orchestration, the current tools in the market, issues related to containers and its orchestration, their solutions as explored by various authors. It also addresses issues related to VM placement in general and how it can be related to container placement over distributed environments.

2 RELATED WORKS

Several surveys have been publishes in the field of optimizing costing of resources and efficient allocation of resources to containers and jobs dynamically fulfilling application constraints and QoS requirements. The following implementations deal with the issues within the container orchestration domain such as – placement of containers, container scheduling, container migration, resource management etc. Some of them are industry level implementaions eg. Kubernetes [1], Docker Swarm [1]. Kubernetes [1] is Google's answer to container orchestration derived from Google's in-house Borg Software. Being open-source, Kubernetes is capable of automating deployment, management and scaling of containerized applications. It defines a cluster of containers as pods which can be found on a node. A node represents a single machine in a cluster. A workload is distributed among the nodes dynamically, as the nodes join or leave the cluster. Kubernetes has automated rollouts and rollbacks to changes made to the applications and it also has a self-healing feature to restart failed containers, replace and reschedule containers when nodes die. Kubernetes provides its API which abstracts all the underlying functions of the orchestrator. Applications can be scaled up or down using simple commands. The API standardizes the cloud providers by abstracting the underlying infrastructure. Docker Swarm [1] is the native tool for clustering docker containers. Service discovery and container scheduling on hosts based upon predefined techniques and constraints are the two basic features Swarm provides. It supports two simple strategies: i) a spread strategy that favors running new containers on least loaded hosts, and ii) a bin packing strategy that favors the most packed hosts that has ample resources to run the containers. Docker Swarm also makes use of a filtering mechanism to identify theQualified resources; users can define filters regarding host statuses (such as available resources) and health check results and container configurations (such as resource and file affinities and dependencies to other containers). Apache Mesos [1] offers cluster management services which includes isolation of resources and shared resources across applications distributed in the cluster. Mesos's job is to insure that there is access to the resources required by the applications within a cluster. The job of Mesos is to insure that applications have access to the resources they require within a cluster. It permits a shared pool of servers that are able to handle distinct units of an application in an independent manner and have the potential to dynamically allocate resources. Mesos is a data center kernel. Chronos scheduler runs on top of Mesos. Mesophere is developed upon the cluster management capabilities of Apache Mesos. Hashicorps' Nomad, Engineered for microservices and batch workloads, its cluster manager and scheduler scale multiple nodes covering datacentres across regions. One advantage of Nomad over Kubernetes and other orchestration tools is that complex applications can be easily expressed without have to care about individual containers that constitute the application. While other orchestration tools offer services like discovery, monitoring secrets management etc., Nomad practices the UNIX tradition of keeping a small scope i.e., focusing on one thing and doing it well and so it only provides scheduling and cluster management. But it does includes tools for service discovery and secret management like Consul and Vault respectively. Nomad is simpler and combines a lightweight scheduler and resource manager into single system. A key difference between Kubernetes and Nomad is that, Kubernetes is open source while Nomad is not which makes Kubernetes a better option for beginners. Guerrero et al, [2] has proposed Non Dominated Sorting Genetic Algorithm II (NSGA over Greedy First fit algorithm to optimize the cost of services for the user, the latency due to migration of microservices in multiple cloud providers. And, the time that any microservice is unavailable due to a failure of one VM, or one provider. They achieved the optimizations by considering the following factors: the VM types and the cloud providers; the number of VM instances for each type; the scale level of each microservices; and the container allocations in the VMs. The solution of NSGA-II was found considerably better than the solution of the greedy algorithm. The Performance can be optimized by using other evolutionary algorithms. Buyya et al, [4] have implemented one simple scheduling algorithm and two autoscaling policies for validation purposes on a prototype by extending the Kubernetes (K8) platform. To schedule a pending pod, the available nodes are first sorted by a random scheduler based upon two factors - remaining resource capacity and amount of resources requested. Once all nodes with capable to execute the pod are discovered, a random one is selected, and the pod submitted to it. If no nodes are found, then the scheduler instructs the autoscaler to scale out. The unschedulable pod is left and an attempt to reschedule is made in the next cycle. Some research areas which remain unexplored are initial placement of containers and application QoS management. Rodriguez et al [5] has put forward a framework and approaches in comparision to Kubernetes scheduler. The paper has addressed the initial placement of containers, autoscaling the number of worker VMs at runtime and a rescheduling mechanism by the integrated use of schedulers, autoscalers, and reschedulers as a mechanism to make orchestration system cloud-aware. The author considers a deployment of a container orchestration framework in a virtualized cloud environment which has access to unlimited number of homogenous VMs. The author further mentions two types of tasks– the first type being long running services that require high availability and handle latency-sensitive requests and the second type being batch jobs which have limited lifetime and are more tolerable towards performance fluctuations. The author has developed custom scheduler which interacts with the Kubernetes API to monitor the state of the pods (group of containers). The results come out to be in favour of the several proposed algorithms against the Kubernetes scheduler in terms of cost. Li et al [15] proposes an optimum minimum migration algorithm to reduce the needless migration of Docker containers in a distributed cloud computing platform. In his paper he discusses Load

imbalance while migrating containers. Li proposes the algorithm by taking into consideration uncertain workloads and changes in data volumes. The algorithm calculates the expansion rate of container volume over a time frame. the algorithm estimates the growth pattern regarding each container and evaluate which container will be moved by sizing the growth rate of Docker containers into the source server.. He refers to these containers as 'Hot containers'. The Hot containers are marked by their relatively big CPU share. Other containers are termed as Cold containers. He points that in traditional minimum migration algorithms, the principle is to sort the container according to their CPU shares in descending order and consequently large containers have the high possibility to be migrated until CPU share of hot servers falls below the threshold and the algorithm does not takes into account the changes in container size. According to Li's algorithm if a container is relatively large, it does get migrated first and does not fall into priority assignment else according to the new arrangement containers with larger memory and faster growth rate have a higher priority to be migrated. Kaewkasi et al [3] this paper presents an improvement in scheduling of the docker containers using Ant colony optimization algorithm. They have implemented their solution by modifying SwarmKit version 4dfc88c. They have compared their ACO scheduler with the original greedy implementation of the scheduler shipped with Swarmkit. Their experimental setup consisted 1 manager node and 5 worker nodes of a Swarmkit cluster on a single DataCenter. According to the ACO scheduling algorithm, an artificial ant randomly looks for resources by looking at the pheromone trail from each resource which is computed by an equation which is the sum of available memory of a node and available CPU of a node multiplied by memory weight and CPU weight respectively. A plain NGINX server with 0.5 core CPU and 128 MB of memory reservation is chosen where 12 NGINX tasks are deployed over 5 nodes. Peinl et al. [7] presents a study on Docker cluster management for the cloud. He provides a case description of a SCHub (Social Collaboration Hub, funded by the BMBF as part of the FHprofUnt funding, https://www.sc-hub.de) project whose aim is to develop a distribution-like collaboration solution based on open source software (OSS) that provides end-users with a consistent experience across all systems while using a modular microservice approach. He has enlisted possible requirements for a container management solution. This paper provides an extensive study on Docker – its dependencies, solutions it provides, the requirements it can fulfil, its alternatives. The paper evaluates some points regarding cluster management solution for Docker and mentions the fact that managing customer data is the most missing part which is the most economic part of the cloud supply chain. Nardelli et al. [6] talks about provisioning of virtual machines in an elastic manner for container deployments. He presented a formulation of EVCD as an Integer Linear Programming problem which considers the heterogeneity of container needs and VM resources. He has compared with two other strategies which are greedy first fit and round robin. The deployment strategies pursued by greedy first fit and round robin heuristics are evaluated by taking EVCD as benchmark. It is found that round robin heuristic is unable to provision virtual machines elastically. Fan et al [11] presents a locality live migration model for docker containers based on load balancing which tries to reduce the amount of communication data in the network without increasing the number of migrations. There are two algorithms proposed for resource selection based on communication distance and live migration of containers respectively. The experimental results depict that the proposed algorithm along with VectorDot load balancing algorithm take less migration time than RIAL algorithm.

TABLE 1. *COMPARATIVE ANALYSIS OF CONTAINER ORCHESTRATION METHODOLOGIES*

Reference	*Objective*	*Environment*	*Technique / Algorithm*	*Results*
Heilig et al. [14]	*Resource management in Multi Cloud environment*	*Java based simulation*	*Biased Random-Key Genetic Algorithm*	*Algorithm proves to be highly competitive against other approaches*
Kim et al.[17]	*TOSCA based and Federation aware cloud orchestration for kubernetes*	*Single Datacenter with 5 VM nodes*	*Ant colony optimization over greedy algorithm*	*14.80% gain than greedy algorithm*
Guerrero et al.[2]	*Optimization of deployment of microservices based applications*	*Custom simulator written in Python*	*NSGA II – Non dominated sorting genetic algorithm*	*Overall improvemen of 300% over greedy algorithm*
Adam et al. [13]	*Stochastic Resource provisioning for containerized multi-tier web services*	*Simulated Environement*	*Two stage stochastic programming resource allocator (2SPRA)*	*Works better than state of the art heuristics*

Kaewkasi et al. [3]	Improvement in container scheduling	Single Datacenter with 5 VM nodes	Ant colony optimization over greedy algorithm	14.80% gain than greedy algorithm
Rodriguez et al. [5]	Initial Placement, autoscaling and rescheduling of containers	Virtualized Public Cloud	Custom algorithm with extended Kubernetes scheduler; custom scheduler	Algorithms work Efficiently than general binpacking methodologies and autoscalers
Hoenisch et al. [12]	Formulating scaling decision as multi-objective optimization problem	Simulated Environement	Custom Algorithm	Reduces average cost per request by 20-28%
Fan et al. [11]	Live migration of containers based on resource locality	A cluster of Physically available servers	Propsoe their own algorithm With comparision to VectorDot and RIAL algorithms	The proposed algorithm performs better than RIAL algorithm
Cortes et al. [10]	combine the different capabilities of the container platforms, with task-based parallel programming models.	Real cloud environment managed by OpenStack over KVM and Bare metal machine	Optimized Combined Algorithms	Has drawbacks compared to other heuristics
Buyya et al.[4]	Conatiner autoschedulin g and autoscaling	Extended Kubernetes (K8)	Custom auto-scheduler	Improved costing and timing values
Li et al.[15]	Reduce the unnecessary migration of Docker containers.	Distributed cloud platform	Optimized minimal migration algorithm	Reduces total migration time, improves resource utilization, realizes load balancing
Nardelli et al. [6]	Elastic provisioning of virtual machines for deploying containers	CPLEX (simulated environment)	Integer linear programming	Optimal solution is obtained as compared to greedy first fit and round robin optimizing different Qos Metrics
Peinl et al. [7]	Docker Cluster management for the cloud	Apache Mesos	Small applications and extensions	Not all requirements fulfilled.
Liang et al. [9]	Container placement and container reassignment	Experiments done in Baidu's data centers,	Communication Aware worst fit decreasing algorithm (CA-WFD)	90% increase in overall throughput
H.Nie et al. [8]	Live Container migration in cloud	High physical configuration servers	Optimized Pre-Copy algorithm, Grey Markov prediction model	Algorithms work Efficiently than general methodologies

3 CONTAINER ORCHESTRATION

When applications are developed in a way in which they form a network of modular services and are independently deployable and where each service operates on a different process and communicates over a lightweight mechanism to meet an objective are termed as microservices and also known as service oriented architecture (SOA). This kind of architecture is now widely adopted in the industry with the help of containers. Containers are lightweight Virtualisation technology primarily available in Linux based systems. When a container based system scales, some major jobs which arise are:

1. Scheduling of containers
2. Management of life cycle

3. Service Discovery

4. LOAD BALANCING

The process of automating deployment of multiple containers, mostly in large numbers, to implement an application is termed as container orchestration. The Docker Command Line Interface (CLI) allows certain actions such as –

1. Pulling a container image from registry
2. Running a container and affixing a terminal to it
3. Assigning the container to a fresh image
4. Uploading image to registry
5. Terminating a running container

But these are limited to a single host running a container. The problem arises when a system scales to multiple containers across distributed multiple hosts. Here it becomes crucial to use a container orchestrator. The term 'Orchestration' is referred extensively to cluster management, container scheduling and provisioning of hosts. Cluster management indicates attaching and detaching of hosts from a cluster, acquiring status information of hosts and containers, and launching and termination of processes. Additionaly a container orchestration software provides, resource limit control, load balancing, health check, fault tolerance and autoscaling[16]. Scheduling describes the scheme to place containers on nodes at a given time instant. Most container schedulers place containers on the basis of resource constraints, node affinity, or both, but more advanced methods can be utilized. The cluster management has access to each host in a cluster which is essential for scheduling services. A scheduler is in charge for automatically finding a host which complies with the provided constraints. Container distribution strategies used by the some popular container providers (eg. Docker, Kubernetes) [11]

1. Random
2. High Availability
3. Emptiest node first (ENF)
4. Binpacking

Load Balancing is an essential part of any orchestration system which manages distribution of load among container instances. The default policy for load balancing is round-robin, but other complex strategies can be used. Fault Tolerance is another major concern for customer facing applications. It is required for an application to be highly available during intense load variations. Health checks determine when a container has to be terminated and restarted. With variation in load, the application is autoscaled by adding or removing containers. The autoscaler works based on a threshold value on CPU or memory. Sophisticated autoscalers can be externally plugged in into the orchestration softwares.

4 CONCLUSIONS

This paper studies the numerous strategies of dealing with containers and its underlying problems. A significant amount of effort has been made in this field and more needs to be explored. Container orchestration is still in its prime years. Technologies like Kubernetes, Mesos, Docker, and Swarm provide the bare minimum services to run a few thousand containers across multiple data centers, but as more companies drift towards these technologies, the requirements start to vary. Application QoS aware orchestration is still in its dormant state, the monitoring of containers is a challenge, security is one of the concerns and so on.

5 REFERENCES

[1] M.A Rodriguez, R. Buyya, "Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions", July 2018, Available at: https://arxiv.org/abs/1807.06193, (2018).

[2] C. Guerrero, I. Lera., and C. Juiz., "Genetic Algorithm for Multi-Objective Optimization of Container Allocation in Cloud Architecture", J. Grid Computing, Vol 16, Issue 1, pp. 113–135, (2018), doi: 10.1109/KST.2017.7886112.

[3] C. Kaewkasi., K. Chuenmuneewong, "Improvement of container scheduling for Docker using Ant Colony Optimization." Proc. Int. Conf. Knowledge and Smart Technology (KST), Thailand, pp. 254-259, (2017), doi: 10.1109/KST.2017.7886112

[4] R. Buyya., M.A Rodriguez, A. Toosi, J. Park., "Cost-Efficient Orchestration of Containers in Clouds: A Vision, Architectural Elements, and Future Directions." J. Physics: Conference Series, Vol 1108, pp, 012001, (Nov 2018)

[5] R. Buyya, M.A Rodriguez, "Containers Orchestration with Cost-Efficient Autoscaling in Cloud Computing Environments", Available at: https://arxiv.org/abs/1812.00300, (Jan 2018).

[6] M. Nardelli, C. Hochreiner, and S. Schulte, "Elastic Provisioning of Virtual Machines for Container Deployment". Proc ACM/SPEC on Int. Conf. on Performance Engineering Companion - ICPE 17 Companion, Italy, pp. 5-10. (2017), doi: 10.1145/3053600.3053602

[7] R. Peinl, F. Holzschuher, F. Pfitzer, "Docker Cluster Management for the Cloud - Survey Results and Own Solution", J. of Grid Computing 14, pp. 265-282, (2016), doi: 10.1007/s10723-016-9366-y

[8] H. Nie, P. Li, H. Xu, L. Dung, J. Song, R. Wang, "Research on Optimized Pre-copy Algorithm of Live Container Migration in Cloud Environment". Communications in Computer and Information Science Parallel Architecture, Algorithm and Programming, Springer Singapore, pp. 554-565, (2017).

[9] L. Lv, Y. Zhang, Y. Li, K. Xu, D. Wang, W. Wang, M. Li, X. Cao, Q. Liang, "Communication-Aware Container Placement and Reassignment in Large-

Scale Internet Data Centers",IEEE J. of Selected Areas of Communication, pp. 540 – 555, (2019), doi: 10.1109/JSAC.2019.2895473

[10] C. Ramon-Cortes, A. Serven, J. Ejarque, D. Lezzi, R. Badia, "Transparent Orchestration of Task-based Parallel Applications in Container Platforms", J. of Grid Computing Vol 16, Issue 1, pp. 137-160, (March 2018), doi: 10.1007/s10723-017-9425-z

[11] W. Fan, Z. Han, P. Li, J. Zhou, J. Fan, R. Wang, "A Live Migration Algorithm for Containers Based on Resource Locality", in J. Signal Processing Systems, (Aug 2018), doi: 10.1007/s11265-018-1401-8.

[12] P. Hoenisch, I. Weber, S. Shulte, L. Zhu, A. Fekete, "Four- Fold Auto-Scaling on a Contemporary Deployment Platform Using Docker Containers" in Service-Oriented Computing Vol 9435, Springer Berlin Heidelberg Publisher, Berlin, pp. 316-323, (2015)

[13] O. Adam, Y.C. Lee, A. Zomaya, "Stochastic Resource Provisioning for Containerized Multi-Tier Web Services in Clouds", in IEEE Trans. on Parallel and Distributed Systems Vol 28 Issue 7pp. 2060-2073, (Jul 2017), doi: 10.1109/TPDS.2016.2639009

[14] L. Heilig, E. Lalla-Ruiz, S. Voß, "A cloud brokerage approach for solving the resource management problem in multi-cloud environments", Computers & Industrial Engineering Vol 95, pp. 16-26, (May 2016), doi: 10.1016/j.cie.2016.02.015

[15] P. Li, H. Nie, He Xu., Lu, D., "A Minimum-Aware Container Live Migration Algorithm in the Cloud Environment", Int. J. Bussiness Data Communication. Network Vol 13 Issue 2, pp. 15-27, (Jun 2017), doi: 10.4018/ijbdcn.2017070102.

[16] E. Casalicchio, "Container Orchestration: A Survey" Systems Modeling: Methodologies and Tools, Springer, pp. 221-235, (2019).

[17] D. Kim, H. Muhammad, E. Kim, S. Helal, C. Lee, "TOSCA Based Federation Aware Cloud Orchestration for Kubernetes Container Platform", Applied Sciences Publicat ions Vol 9 Issue 1, pp. 191, (2019), doi: 10.3390/app9010191