The University of Melbourne
School of Computing and Information Systems
COMP90044 Research Methods

The State of the Art in Stateful Orchestration

834063

Introduction

Distributed architectures have seen a revolution over the past decade, allowing for software to be composed from loosely coupled and independent services at a much larger scale. Driving this shift is a change in how resources are virtualised and provided to applications. Namely the shift from kernel virtualisation implemented commonly in a virtual machine or VM, where a hypervisor brokers resources at a simulated hardware layer with substantial overhead [6], to more lightweight methods such as application containers [19]. Containers provide a layer of low cost abstraction by packaging dependencies, configuration and applications into a single unit, whilst maintaining a smaller storage footprint than a VM image. Containers moreover provide less cumbersome process isolation, with recent studies showing measurable decreases in resource overhead, startup and shutdown times over kernel virtualisation (VMs) [10] [16]. Container's success is also attributed to an easier management of the distributed applications life cycle [13] and application portability.

Containers are isolated by design, requiring configuration to communicate between containers. Containers are also commonly deployed as a composition of hundreds to thousands of independent services[16]. Hence, managing container oriented architectures for larger applications necessitates an organized framework, where an application can be may be scaled over several thousand units of compute according to requirements, service level agreements (SLA) and quality of service targets (QoS). This framework acts to provide an API for selecting, deploying, monitoring and dynamically controlling containers [19].The most popular framework currently in use is Kubernetes [18], a full featured orchestration middle-ware. Docker Swarm and Apache Mesos are alternatives with different design approaches, however much less frequently used [14].

Stateless services within an architecture can be easily replicated as they perform as functional units. An orchestrator can load balance traffic from one stateless service to another as they are functionally interchangeable, holding homogeneous (no) state. Stateful services however cannot. Stateful services instead have their state transitioned as a result of the events processed during run-time, meaning that without first replicating the events, it is not (easily) possible to interchange these services without varying levels of disruption [17]. Fortunately this inconvenience can be avoided for many modern asynchronous applications without real time requirements. State can be stored in a persistent store, or semi-persistent store in these use cases such as Redis and alike. Applications with strict real time requirements however, are unable to benefit fully from external data stores and instead are required to maintain a greater degree of state in volatile memory. Popular examples of applications with these real-time, stateful requirements are massive online multiplayer games (mmo) and streaming services.

Deploying stateful services requires further coordination when introducing redundancy into the system, replicas must be synchronized whilst overhead complexity is increased. In order to benefit from the aforementioned gains of containers at scale whilst maintaining QoS and SLAs it is necessary that the orchestrator addresses the unique requirements imposed.

Background

At the core of container technology, there are cgroups [2] and Linux namespace [5]. Containers can be broadly split into application and system containers. A system container runs a full operating system, while an application container provides a lightweight abstraction composed of multiple software layers, each providing a subset of the entire containers functionality. Allowing for caching and shared layers among containers and reducing network and storage costs.

Container orchestrators offer resource limit control, scheduling, load balancing, health checking, fault tolerance and auto scaling. Resource limit control allows CPU and memory reservation for specified containers, as opposed to traditional shared resource allocation by request. Scheduling allows declarative configuration of containers, compute and storage according real time demand, resource constraints and external requirements. Load balancing assigns request among suitable container instances, with a default round-robin policy. The health check informs the orchestration engine whether a container is able to respond to requests. Fault tolerance is achieved via high availability and replication controllers which allow control over horizontal scaling of containers and target redundancy. Auto scaling defines a policy to automatically include and remove containers, typically the policies are defined in terms of resource thresholds such as CPU and memory however there has been a shift towards defining QoS based policies [24] [9].

Literature Selection

To develop a deeper understanding of the current state of the art in stateful orchestration, this review will will discuss past literature in two categories.

- Stateful replication and partition tolerance of state sets.
- Planning models of container based systems, entailing constraints imposed by functional and non-functional requirements

These studies will be analyzed to synthesize future directions, identify key performance metrics and applicable methods of broadly related research to stateful orchestration; with an emphasis on Kuberenetes as the orchestrator of choice due to its large scale adoption.

Discussion

Stateful Replication and Fault Tolerance

Availability is a focus of infrastructure, software and service providers alike. There has been extensive research conducted in the areas of partition tolerance in distributed systems, pioneered in early research [4][1][3]. Within orchestration, Kubernetes addresses availability via self the healing capability which restarts or schedules new container(s) in the event of a crash failure. Although self healing improves availability of services, redundancy is still the primary instrument to facilitate highly available applications in a orchestrated environment [20].

Netto et al [13] published early research into methods that increased partition tolerance among stateful distributed applications, orchestrated by Kuberenetes. This early work identified issues surrounding stateful replication, namely the large overhead incurred. The authors examine a logging server, with strict ordering and reliability requirements as the experiment application. To instrument partition tolerance, replicated containers are introduced

behind a proxy. To enforce the state replication between containers, the authors introduce DORADO [13] a protocol for leader elected message passing. In summary, in DORADO the leader writes the ordering of requests to etcd, the distributed data storage shipped with Kuberenetes. Following this, all other replicated containers execute the requests in the order defined within etcd, whilst only the leader responds to the request. This achieves the target redundancy via leader election in the event of a failure with replicated containers maintaining homogeneous state. Despite the papers focus on availability, the experiments conducted instead emphasize the overhead effects of replication on response latency with respect to replication factor achieved as opposed to availability metrics such as relative downtime and the time taken to recover from leader crash.

Koordinator [15] follows in a similar suit, this time as a service deployed alongside Kuberenetes as opposed to tighter integration shown in [13]. This implementation proposes a firewall solution, where all requests are routed via the implemented Koordinator service which then multicasts messages to available containers. However as Casalicchio et al notes [19], the Koordinator firewall that routes requests remains as a single point of failure despite the benefits of state replication. [15]. The methods discussed in [15] follow in a similar paradigm to [13] to achieve target redundancy, via message and therefore state replication between containers.

In both [13] [15] the testing environment consisted of 4 compute nodes. For many modern cloud applications, this level of distribution is unrealistically low compared to the many hundreds or thousands of containers deployed at scale in mature use cases [8]. Likewise both papers used an example logging server in their experiment, where request ordering is more important that request latency. More specifically, ordering by the etcd's Raft algorithm [25] within DORADO and total ordering in Koordinator remained the focus of both papers, as opposed to low-latency replication. In contrast, the examples of real time highly distributed applications previously mentioned (mmo, streaming) a delayed request becomes quickly invalidated, thus imposing stricter requirements on delivery speed.[12]

In [19] Casalicchio et al proposes a novel secondary tag based approach, managed by a per-container state controller. The authors in particular aim to take a Kuberenetes integrated approach, targeting two separate controllers: Stateful Set and Deployment. This approach offers a native approach to tackling state replication within Kuberenetes orchestrated applications as opposed to [15] [13]. In similar vein the experiment settings entail less realistic distribution, with 4 VMs in use.The experiment application, a video on demand streaming service does have real time requirements, storing per tenant state for marking video progression. The authors also present their results with consideration of widely adopted metrics such as the time taken to relative downtime and recovery delay from a fault, providing a more realistic insight into partition tolerance then previously mentioned papers.

The start of the art in stateful replication is primarily concerned with trade offs between performance overhead and replication, high availability via integrated, service and interception controllers. Potential research areas include incorporating elastic demand scaling, as all aforementioned papers conduct experiments with static replication targets. Likewise, further research can be conducted to explore larger scale workflows, in particular identifying possible scaling constraints on existing replication methods discussed.

Planning

Planning for optimal resource allocation is a prominent area of research within cloud computing [7]. Literature has matured over the past five years, moving away from traditional infrastructure level indicators such as CPU and memory utilization. Instead focus is shifting to QoS, cost and service level indicators. In particular, auto-scaler and resource scheduler

methods have been investigated with aims of reducing cost whilst maintaining QoS guarantees. Stateful orchestration can be considered a subset of this area of research, with respect to the unique QoS and resource requirements imposed by the previously mentioned use cases. The following discussion will focus on methods and outcomes which can shape the future directions within the younger paradigm of stateful orchestration.

Early research in orchestration planning by Al-Dhuraibi et al [11] a proposed vertical vertical preferences scaling algorithm for the Docker Swarm auto-scaler, named ElasticDocker. ElasticDocker's auto-scaler first considers resource allocation locally (vertical) and, if unable will live-migrate the application in order to meet workload resource demand. This early literature laid the foundation for future authors discussion on default orchestrator scaling algorithms.

Rodriguez et al [21] propose an integrated module for cost efficient resource allocation, within the Kubernetes orchestrator. The authors integrate the use of auto-scaler, scheduler and re-schedulers to achieve cost reduction whilst maintaining QoS. This integrated approach demonstrated the compounding benefits of multiple tailored orchestrator components over the default Kuberenetes offering. Moreover highlighting the extent to which Kubernetes' generality as a platform in turn leaves measurable room for application specific improvement. This paper however does not address optimization for heterogeneous application requirements. In particular addressing resource consumption estimation and dichotomy of long versus short running jobs. In similar focus, Truyen et al [22] integrates tenant SLA and QoS across a multi-cloud, heterogeneous infrastructure environment. Truyen et al's proposed solution addresses the insufficiency of infrastructure metrics in orchestration planning.

The authors in [26] address heterogeneous workflows and short versus long running application lifetimes, within the lens of QoS management policies, as opposed to infrastructure level metrics [26]. The authors propose a scheduling component HTAS, to support heterogeneity between long running applications and short lived jobs. The authors work addresses the gaps in [21], resulting in between 23% and 32% decrease in cost over default Kubernetes and a selected, non orchestrated alternative. Another relevant method to stateful orchestration proposed by the authors in [26] is the live migration and check-pointing method for application reallocation. This method was used by the authors in order to optimize cost efficiency when underutilized compute nodes were shut down. To facilitate this, CRIU [23] was employed for container and memory live capture [23]. This proposed approach, demonstrated an alternative to the previously discussed replication methods for stateful containers. The network bandwidth and migration time required appear however infeasible under current cost constraints for long lived applications[26]. The live migration aided by CRIU is not a new concept to in cloud computing and was introduced in earlier research for containerized applications [23] [11].

Conclusion

This paper identified literature on availability through replication and planning in regards cost, QoS and requirements as areas of interest. The current literature on stateful replication and fault tolerance is focused on examining and improving upon the options that orchestrators provide in configuring and planning redundancy for highly available systems. Challenges within this area are remain the unrealistic and static current state of experiments, where elasticity and dynamic auto-scaling of solutions is required to bridge the gap to mainstream use cases.

Planning challenges are investigated by research on run-time adaptions, implemented primarily via the auto-scaler and scheduler. Different approaches were taken to optimise

cost, with constraints focused firmly within QoS as opposed to traditional infrastructure metrics. Further, the adaptation for cross-cloud and heterogeneous resources in order to meet similarly non-homogeneous workflows showed increasing maturity within the literature reviewed. Within auto-scaling solutions discussed, vertical and horizontal were considered, however a hybrid approach also displayed the most efficacy. In order to support this hybrid approach, solutions considered pre-existing live migration tooling and explored its viability within Kubernetes.

Future Directions

Stateful orchestration of containers is still in its relative infancy, whilst planning and more non-stateful orchestration literature is relatively mature. Further research needs to be conducted to address: (1) application demand elasticity whilst preserving high availability. (2) Examine the applicability of existing planning methods discussed as they relate to stateful orchestration. (3) Investigate existing platforms for stateful application management and interrogate the performance of orchestration in comparison to these platforms with regards to cost, complexity and availability.

References

[1] Ernst Schmitter. "FAULT TOLERANCE IN DISTRIBUTED SYSTEMS." In: Siemens Forschungs- und Entwicklungsberichte/Siemens Research and Development Reports (1983). issn: 03709736.

[2] Rob Pike et al. "The use of name spaces in plan 9". In: Proceedings of the 5th ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring, EW 1992. 1992. doi: 10.1145/506378.506413.

[3] Rachid Guerraoui and Andre Schiper. "Fault-tolerance by replication in distributed systems". In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1996. isbn: 354061317X. doi: 10.1007/BFb0013477.

[4] Leslie Lamport. "The Part-Time Parliament". In: ACM Transactions on Computer Systems (1998). issn: 07342071. doi: 10.1145/279227.279229.

[5] Eric W Biederman. "Multiple Instances of the Global Linux Namespaces". In: Linux Symposium Volume One (2006). doi: 10.1.1.108.5475.

[6] Miguel G. Xavier et al. "Performance evaluation of container-based virtualization for high performance computing environments". In: Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013. 2013. isbn: 9780769549392. doi: 10.1109/PDP.2013.41.

[7] Sunilkumar S. Manvi and Gopal Krishna Shyam. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. 2014. doi: 10.1016/j.jnca.2013.10.004.

[8] Brendan Burns et al. "Borg, omega, and kubernetes". In: Communications of the ACM (2016). issn: 15577317. doi: 10.1145/2890784.

[9] Emiliano Casalicchio and Vanessa Perciballi. "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics". In: Proceedings - 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2017. 2017. isbn: 9781509065585. doi: 10.1109/FAS-W.2017.149.

[10] Emiliano Casalicchio and Vanessa Perciballi. "Measuring Docker performance: What a mess!!!" In: ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering. 2017. isbn: 9781450348997. doi: 10.1145/3053600.3053605.

[11] Yahya Al-Dhuraibi et al. "Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER". In: IEEE International Conference on Cloud Computing, CLOUD. 2017. isbn: 9781538619933. doi: 10.1109/CLOUD.2017.67.

[12] Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems. 2017. isbn: 9781491903063.

[13] Hylson V. Netto et al. "State machine replication in containers managed by Kubernetes". In: Journal of Systems Architecture (2017). issn: 13837621. doi: 10.1016/j.sysarc.2016.12.007.

[14] Claus Pahl et al. "Cloud Container Technologies: a State-of-the-Art Review". In: IEEE Transactions on Cloud Computing (May 2017). issn: 21687161. doi: 10.1109/TCC.2017.2702586.

[15] Hylson Vescovi Netto et al. "Koordinator: A Service Approach for Replicating Docker Containers in Kubernetes". In: Proceedings - IEEE Symposium on Computers and Communications. 2018. isbn: 9781538669501. doi: 10.1109/ISCC.2018.8538452.

[16] Pankaj Saha et al. "Evaluation of docker containers for scientific workloads in the cloud". In: ACM International Conference Proceeding Series. 2018. isbn: 9781450364461. doi: 10.1145/3219104.3229280.

[17] Leila Abdollahi Vayghan et al. "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes". In: Proceedings - 19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019. 2019. isbn: 9781728139272. doi: 10.1109/QRS.2019.00034.

[18] Isam Mashhour Al Jawarneh et al. "Container Orchestration Engines: A Thorough Functional and Performance Comparison". In: IEEE International Conference on Communications. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., May 2019. isbn: 9781538680889. doi: 10.1109/ICC.2019.8762053.

[19] Emiliano Casalicchio. "Container Orchestration: A Survey". In: Springer, Cham, 2019, pp. 221–235. doi: 10.1007/978-3-319-92378-914.

[20] Daniel Ford et al. "Availability in globally distributed storage systems". In: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010. 2019. isbn: 9781931971799.

[21] Maria A. Rodriguez and Rajkumar Buyya. "Container-based cluster orchestration systems: A taxonomy and future directions". In: Software: Practice and Experience 49.5 (May 2019), pp. 698–719. issn: 0038-0644. doi: 10.1002/spe.2660. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2660.

[22] Eddy Truyen et al. "A comprehensive feature comparison study of open-source container orchestration frameworks". In: Applied Sciences (Switzerland) 9.5 (Feb. 2019). issn: 20763417. doi: 10.3390/app9050931. url: http://arxiv.org/abs/2002.02806.

[23] Ranjan Sarpangala Venkatesh et al. "Fast in-memory CRIU for docker containers". In: ACM International Conference Proceeding Series. 2019. isbn: 9781450372060. doi: 10.1145/3357526.3357542.

[24] Emiliano Casalicchio and Stefano Iannucci. "The state-of-the-art in container technologies: Application, orchestration and security". In: Concurrency Computation. 2020. doi: 10.1002/cpe.5668.

[25] Lars Larsson et al. "Impact of etcd Deployment on Kubernetes, Istio, and Application Performance". In: (Mar. 2020). url: http://arxiv.org/abs/2004.00372.

[26] Zhiheng Zhong and Rajkumar Buyya. "A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources". In: ACM Transactions on Internet Technology 20.2 (2020), p. 15. issn: 15576051. doi: 10.1145/3378447. url: https://doi.org/10.1145/3378447.